ChibiOS/NIL

4.1.3

Reference Manual

Sat Sep 13 2025 08:11:16

Chapter 1

Introduction

Author

Giovanni Di Sirio (gdisirio@users.sourceforge.net).

Why Nil?

Of course because it is so small that it is almost nil. I wrote Nil because I wanted to experiment with an idea I had regarding a minimal RTOS. Basically I wanted to verify how small could be an RTOS while retaining useful features.

Features

- Free software, GPL3 licensed. Stable releases include a exception clause to the GPL.
- Designed for realtime applications.
- Easily portable.
- Preemptive scheduling.
- Each thread has its own priority level.
- Offers tasks, time, semaphores, event flags, timeouts.
- Fully static.
- Minimal system requirements: about 700 bytes ROM with all options enabled.
- Almost totally written in C with little ASM code required for ports.
- Compatible with ChibiOS/HAL.
- API compatible with ChibiOS/RT of which, ChibiOS/NIL, is functionally a subset.

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Topic Documentation

6.1 Compiler portability.

6.1.1 Detailed Description

Compiler abstraction macros

- #define CC_SECTION(s)

  Allocates a variable or function to a specific section.
- #define CC_WEAK __attribute__((weak))

  Marks a function or variable as a weak symbol.
- #define CC_USED __attribute__((used))

  Marks a function or variable as used.
- #define CC_ALIGN_DATA(n)

  Enforces alignment of the variable declared afterward.
- #define CC_ALIGN_CODE(n)

  Enforces alignment of a function declared afterward.
- #define CC_PACK __attribute__((packed))

  Enforces packing of the structure declared afterward.
- #define CC_NO_INLINE __attribute__((noinline))

  Marks a function as not inlineable.
- #define CC_FORCE_INLINE __attribute__((always_inline))

  Enforces a function inline.
- #define CC_NO_RETURN __attribute__((noreturn))

  Marks a function as non-returning.
- #define CC_ROMCONST const

  Enforces a variable in a ROM area.
- #define CC_LIKELY(x)

  Marks a boolean expression as likely true.
- #define CC_UNLIKELY(x)

  Marks a boolean expression as likely false.

6.1.2 Macro Definition Documentation

6.1.2.1 CC_SECTION

```
#define CC_SECTION(
                s)
```

Value:

```
__attribute__((section(s)))
```

Allocates a variable or function to a specific section.

Note

If the compiler does not support such a feature then this macro must not be defined or it could originate errors.

6.1.2.2 CC_WEAK

```
#define CC_WEAK __attribute__((weak))
```

Marks a function or variable as a weak symbol.

Note

If the compiler does not support such a feature then this macro must not be defined or it could originate errors.

6.1.2.3 CC_USED

```
#define CC_USED __attribute__((used))
```

Marks a function or variable as used.

The compiler or linker shall not remove the marked function or variable regardless if it is referred or not in the code.

Note

If the compiler does not support such a feature then this macro must not be defined or it could originate errors.

6.1.2.4 CC_ALIGN_DATA

```
#define CC_ALIGN_DATA(
                n)
```

Value:

```
__attribute__((aligned(n)))
```

Enforces alignment of the variable declared afterward.

Note

If the compiler does not support such a feature then this macro must not be defined or it could originate errors.

6.1.2.5 CC_ALIGN_CODE

```
#define CC_ALIGN_CODE(
              n)
```

Value:

```
__attribute__((aligned(n)))
```

Enforces alignment of a function declared afterward.

Note

If the compiler does not support such a feature then this macro must not be defined or it could originate errors.

6.1.2.6 CC_PACK

```
#define CC_PACK __attribute__((packed))
```

Enforces packing of the structure declared afterward.

Note

If the compiler does not support such a feature then this macro must not be defined or it could originate errors.

6.1.2.7 CC_NO_INLINE

```
#define CC_NO_INLINE __attribute__((noinline))
```

Marks a function as not inlineable.

Note

Can be implemented as an empty macro if not supported by the compiler.

6.1.2.8 CC_FORCE_INLINE

```
#define CC_FORCE_INLINE __attribute__((always_inline))
```

Enforces a function inline.

Note

Can be implemented as an empty macro if not supported by the compiler.

6.1.2.9 CC_NO_RETURN

```
#define CC_NO_RETURN __attribute__((noreturn))
```

Marks a function as non-returning.

Note

Can be implemented as an empty macro if not supported by the compiler.

6.1.2.10 CC_ROMCONST

```
#define CC_ROMCONST const
```

Enforces a variable in a ROM area.

Note

Can be implemented as an empty macro if not supported by the compiler.

6.1.2.11 CC_LIKELY

```
#define CC_LIKELY(
              x)
```

Value:

```
__builtin_expect(!!(x), 1)
```

Marks a boolean expression as likely true.

Parameters

in	*x*	a valid expression

6.1.2.12 CC_UNLIKELY

```
#define CC_UNLIKELY(
              x)
```

Value:

```
__builtin_expect(!!(x), 0)
```

Marks a boolean expression as likely false.

Parameters

in	*x*	a valid expression

6.2 NIL Kernel

6.2.1 Detailed Description

The kernel is the portable part of ChibiOS/NIL, this section documents the various kernel subsystems.

Topics

- Configuration
- Base API
- Semaphores
- Events
- Synchronous Messages

6.2.2 Configuration

6.2.2.1 Detailed Description

Kernel related settings and hooks.

Kernel parameters and options

- #define CH_CFG_MAX_THREADS 4

  Maximum number of user threads in the application.
- #define CH_CFG_AUTOSTART_THREADS TRUE

  Auto starts threads when `chSysInit()` is invoked.

System timer settings

- #define CH_CFG_ST_RESOLUTION 32

  System time counter resolution.
- #define CH_CFG_ST_FREQUENCY 1000

  System tick frequency.
- #define CH_CFG_ST_TIMEDELTA 0

  Time delta constant for the tick-less mode.

Subsystem options

- #define CH_CFG_USE_WAITEXIT TRUE

  Threads synchronization APIs.
- #define CH_CFG_USE_SEMAPHORES TRUE

  Semaphores APIs.
- #define CH_CFG_USE_MUTEXES FALSE

  Mutexes APIs.
- #define CH_CFG_USE_EVENTS TRUE

  Events Flags APIs.
- #define CH_CFG_USE_MESSAGES TRUE

  Synchronous Messages APIs.

OSLIB options

- #define CH_CFG_USE_MAILBOXES TRUE

  Mailboxes APIs.
- #define CH_CFG_USE_MEMCORE TRUE

  Core Memory Manager APIs.
- #define CH_CFG_MEMCORE_SIZE 0

  Managed RAM size.
- #define CH_CFG_USE_HEAP TRUE

  Heap Allocator APIs.
- #define CH_CFG_USE_MEMPOOLS TRUE

  Memory Pools Allocator APIs.
- #define CH_CFG_USE_OBJ_FIFOS TRUE

  Objects FIFOs APIs.
- #define CH_CFG_USE_PIPES TRUE

  Pipes APIs.
- #define CH_CFG_USE_OBJ_CACHES TRUE

  Objects Caches APIs.
- #define CH_CFG_USE_DELEGATES TRUE

  Delegate threads APIs.
- #define CH_CFG_USE_JOBS TRUE

  Jobs Queues APIs.

Objects factory options

- #define CH_CFG_USE_FACTORY TRUE

  Objects Factory APIs.
- #define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8

  Maximum length for object names.
- #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

  Enables the registry of generic objects.
- #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

  Enables factory for generic buffers.
- #define CH_CFG_FACTORY_SEMAPHORES TRUE

  Enables factory for semaphores.
- #define CH_CFG_FACTORY_MAILBOXES TRUE

  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_PIPES TRUE

  Enables factory for Pipes.

Debug options

- #define CH_DBG_STATISTICS FALSE

  Debug option, kernel statistics.
- #define CH_DBG_SYSTEM_STATE_CHECK TRUE

  Debug option, system state check.
- #define CH_DBG_ENABLE_CHECKS TRUE

  Debug option, parameters checks.
- #define CH_DBG_ENABLE_ASSERTS TRUE

  System assertions.
- #define CH_DBG_ENABLE_STACK_CHECK TRUE

  Stack check.

Kernel hooks

- #define CH_CFG_SYSTEM_INIT_HOOK()

  System initialization hook.
- #define CH_CFG_THREAD_EXT_FIELDS /* Add threads custom fields here.*/

  Threads descriptor structure extension.
- #define CH_CFG_THREAD_EXT_INIT_HOOK(tr)

  Threads initialization hook.
- #define CH_CFG_THREAD_EXIT_HOOK(tp)

  Threads finalization hook.
- #define CH_CFG_IDLE_ENTER_HOOK()

  Idle thread enter hook.
- #define CH_CFG_IDLE_LEAVE_HOOK()

  Idle thread leave hook.
- #define CH_CFG_SYSTEM_HALT_HOOK(reason)

  System halt hook.

Macros

- #define _CHIBIOS_NIL_CONF_
- #define _CHIBIOS_NIL_CONF_VER_4_0_

6.2.2.2 Macro Definition Documentation

6.2.2.2.1 _CHIBIOS_NIL_CONF_

```
#define _CHIBIOS_NIL_CONF_
```

6.2.2.2.2 _CHIBIOS_NIL_CONF_VER_4_0_

```
#define _CHIBIOS_NIL_CONF_VER_4_0_
```

6.2.2.2.3 CH_CFG_MAX_THREADS

```
#define CH_CFG_MAX_THREADS 4
```

Maximum number of user threads in the application.

Note

This number is not inclusive of the idle thread which is implicitly handled.

Set this value to be exactly equal to the number of threads you will use or you would be wasting RAM and cycles.

This values also defines the number of available priorities (0..CH_CFG_MAX_THREADS-1).

6.2.2.2.4 CH_CFG_AUTOSTART_THREADS

```
#define CH_CFG_AUTOSTART_THREADS TRUE
```

Auto starts threads when `chSysInit()` is invoked.

6.2.2.2.5 CH_CFG_ST_RESOLUTION

```
#define CH_CFG_ST_RESOLUTION 32
```

System time counter resolution.

Note

Allowed values are 16 or 32 bits.

6.2.2.2.6 CH_CFG_ST_FREQUENCY

```
#define CH_CFG_ST_FREQUENCY 1000
```

System tick frequency.

Note

This value together with the `CH_CFG_ST_RESOLUTION` option defines the maximum amount of time allowed for timeouts.

6.2.2.2.7 CH_CFG_ST_TIMEDELTA

```
#define CH_CFG_ST_TIMEDELTA 0
```

Time delta constant for the tick-less mode.

Note

If this value is zero then the system uses the classic periodic tick. This value represents the minimum number of ticks that is safe to specify in a timeout directive. The value one is not valid, timeouts are rounded up to this value.

6.2.2.2.8 CH_CFG_USE_WAITEXIT

```
#define CH_CFG_USE_WAITEXIT TRUE
```

Threads synchronization APIs.

If enabled then the `chThdWait()` function is included in the kernel.

Note

The default is `TRUE`.

6.2.2.2.9 CH_CFG_USE_SEMAPHORES

```
#define CH_CFG_USE_SEMAPHORES TRUE
```

Semaphores APIs.

If enabled then the Semaphores APIs are included in the kernel.

Note

The default is `TRUE`.

6.2.2.2.10 CH_CFG_USE_MUTEXES

```
#define CH_CFG_USE_MUTEXES FALSE
```

Mutexes APIs.

If enabled then the mutexes APIs are included in the kernel.

Note

Feature not currently implemented.
The default is `FALSE`.

6.2.2.2.11 CH_CFG_USE_EVENTS

```
#define CH_CFG_USE_EVENTS TRUE
```

Events Flags APIs.

If enabled then the event flags APIs are included in the kernel.

Note

The default is `TRUE`.

6.2.2.2.12 CH_CFG_USE_MESSAGES

```
#define CH_CFG_USE_MESSAGES TRUE
```

Synchronous Messages APIs.

If enabled then the synchronous messages APIs are included in the kernel.

Note

The default is `TRUE`.

6.2.2.2.13 CH_CFG_USE_MAILBOXES

```
#define CH_CFG_USE_MAILBOXES TRUE
```

Mailboxes APIs.

If enabled then the asynchronous messages (mailboxes) APIs are included in the kernel.

Note

The default is `TRUE`.

Requires `CH_CFG_USE_SEMAPHORES`.

6.2.2.2.14 CH_CFG_USE_MEMCORE

```
#define CH_CFG_USE_MEMCORE TRUE
```

Core Memory Manager APIs.

If enabled then the core memory manager APIs are included in the kernel.

Note

The default is `TRUE`.

6.2.2.2.15 CH_CFG_MEMCORE_SIZE

```
#define CH_CFG_MEMCORE_SIZE 0
```

Managed RAM size.

Size of the RAM area to be managed by the OS. If set to zero then the whole available RAM is used. The core memory is made available to the heap allocator and/or can be used directly through the simplified core memory allocator.

Note

In order to let the OS manage the whole RAM the linker script must provide the `__heap_base__` and `__heap_end__` symbols.

Requires `CH_CFG_USE_MEMCORE`.

6.2.2.2.16 CH_CFG_USE_HEAP

```
#define CH_CFG_USE_HEAP TRUE
```

Heap Allocator APIs.

If enabled then the memory heap allocator APIs are included in the kernel.

Note

The default is `TRUE`.

6.2.2.2.17 CH_CFG_USE_MEMPOOLS

```
#define CH_CFG_USE_MEMPOOLS TRUE
```

Memory Pools Allocator APIs.

If enabled then the memory pools allocator APIs are included in the kernel.

Note

The default is TRUE.

6.2.2.2.18 CH_CFG_USE_OBJ_FIFOS

```
#define CH_CFG_USE_OBJ_FIFOS TRUE
```

Objects FIFOs APIs.

If enabled then the objects FIFOs APIs are included in the kernel.

Note

The default is TRUE.

6.2.2.2.19 CH_CFG_USE_PIPES

```
#define CH_CFG_USE_PIPES TRUE
```

Pipes APIs.

If enabled then the pipes APIs are included in the kernel.

Note

The default is TRUE.

6.2.2.2.20 CH_CFG_USE_OBJ_CACHES

```
#define CH_CFG_USE_OBJ_CACHES TRUE
```

Objects Caches APIs.

If enabled then the objects caches APIs are included in the kernel.

Note

The default is TRUE.

6.2.2.2.21 CH_CFG_USE_DELEGATES

```
#define CH_CFG_USE_DELEGATES TRUE
```

Delegate threads APIs.

If enabled then the delegate threads APIs are included in the kernel.

Note

The default is `TRUE`.

6.2.2.2.22 CH_CFG_USE_JOBS

```
#define CH_CFG_USE_JOBS TRUE
```

Jobs Queues APIs.

If enabled then the jobs queues APIs are included in the kernel.

Note

The default is `TRUE`.

6.2.2.2.23 CH_CFG_USE_FACTORY

```
#define CH_CFG_USE_FACTORY TRUE
```

Objects Factory APIs.

If enabled then the objects factory APIs are included in the kernel.

Note

The default is `FALSE`.

6.2.2.2.24 CH_CFG_FACTORY_MAX_NAMES_LENGTH

```
#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8
```

Maximum length for object names.

If the specified length is zero then the name is stored by pointer but this could have unintended side effects.

6.2.2.2.25 CH_CFG_FACTORY_OBJECTS_REGISTRY

```
#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE
```

Enables the registry of generic objects.

6.2.2.2.26 CH_CFG_FACTORY_GENERIC_BUFFERS

```
#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE
```

Enables factory for generic buffers.

6.2.2.2.27 CH_CFG_FACTORY_SEMAPHORES

```
#define CH_CFG_FACTORY_SEMAPHORES TRUE
```

Enables factory for semaphores.

6.2.2.2.28 CH_CFG_FACTORY_MAILBOXES

```
#define CH_CFG_FACTORY_MAILBOXES TRUE
```

Enables factory for mailboxes.

6.2.2.2.29 CH_CFG_FACTORY_OBJ_FIFOS

```
#define CH_CFG_FACTORY_OBJ_FIFOS TRUE
```

Enables factory for objects FIFOs.

6.2.2.2.30 CH_CFG_FACTORY_PIPES

```
#define CH_CFG_FACTORY_PIPES TRUE
```

Enables factory for Pipes.

6.2.2.2.31 CH_DBG_STATISTICS

```
#define CH_DBG_STATISTICS FALSE
```

Debug option, kernel statistics.

Note

Feature not currently implemented.
The default is `FALSE`.

6.2.2.2.32 CH_DBG_SYSTEM_STATE_CHECK

```
#define CH_DBG_SYSTEM_STATE_CHECK TRUE
```

Debug option, system state check.

Note

The default is `FALSE`.

6.2.2.2.33 CH_DBG_ENABLE_CHECKS

```
#define CH_DBG_ENABLE_CHECKS TRUE
```

Debug option, parameters checks.

Note

The default is FALSE.

6.2.2.2.34 CH_DBG_ENABLE_ASSERTS

```
#define CH_DBG_ENABLE_ASSERTS TRUE
```

System assertions.

Note

The default is FALSE.

6.2.2.2.35 CH_DBG_ENABLE_STACK_CHECK

```
#define CH_DBG_ENABLE_STACK_CHECK TRUE
```

Stack check.

Note

The default is FALSE.

6.2.2.2.36 CH_CFG_SYSTEM_INIT_HOOK

```
#define CH_CFG_SYSTEM_INIT_HOOK()
```

Value:

```
{                                       \
}
```

System initialization hook.

6.2.2.2.37 CH_CFG_THREAD_EXT_FIELDS

```
#define CH_CFG_THREAD_EXT_FIELDS  /* Add threads custom fields here.*/
```

Threads descriptor structure extension.

User fields added to the end of the thread_t structure.

6.2.2.2.38 CH_CFG_THREAD_EXT_INIT_HOOK

```
#define CH_CFG_THREAD_EXT_INIT_HOOK(
              tr)
```

Value:

```
  {                                   \
  /* Add custom threads initialization code here.*/                          \
}
```

Threads initialization hook.

6.2.2.2.39 CH_CFG_THREAD_EXIT_HOOK

```
#define CH_CFG_THREAD_EXIT_HOOK(
              tp)
```

Value:

```
{}
```

Threads finalization hook.

User finalization code added to the `chThdExit()` API.

6.2.2.2.40 CH_CFG_IDLE_ENTER_HOOK

```
#define CH_CFG_IDLE_ENTER_HOOK()
```

Value:

```
{                                         \
}
```

Idle thread enter hook.

Note

This hook is invoked within a critical zone, no OS functions should be invoked from here.

This macro can be used to activate a power saving mode.

6.2.2.2.41 CH_CFG_IDLE_LEAVE_HOOK

```
#define CH_CFG_IDLE_LEAVE_HOOK()
```

Value:

```
{                                         \
}
```

Idle thread leave hook.

Note

This hook is invoked within a critical zone, no OS functions should be invoked from here.

This macro can be used to deactivate a power saving mode.

6.2.2.2.42 CH_CFG_SYSTEM_HALT_HOOK

```
#define CH_CFG_SYSTEM_HALT_HOOK(
              reason)
```

Value:

```
{                                   \
}
```

System halt hook.

6.2.3 Base API

6.2.3.1 Detailed Description

Kernel types

- typedef port_rtcnt_t rtcnt_t
- typedef port_syssts_t syssts_t
- typedef port_stkalign_t stkalign_t
- typedef uint8_t tstate_t
- typedef uint32_t tprio_t
- typedef int32_t msg_t
- typedef int32_t eventid_t
- typedef uint32_t eventmask_t
- typedef uint32_t eventflags_t
- typedef int32_t cnt_t
- typedef uint32_t ucnt_t

ChibiOS/NIL version identification

- #define CH_KERNEL_VERSION "4.1.3"

  Kernel version string.
- #define CH_KERNEL_MAJOR 4

  Kernel version major number.
- #define CH_KERNEL_MINOR 1

  Kernel version minor number.
- #define CH_KERNEL_PATCH 3

  Kernel version patch number.

Constants for configuration options

- #define FALSE 0

  Generic 'false' preprocessor boolean constant.
- #define TRUE 1

  Generic 'true' preprocessor boolean constant.

Wakeup messages

- #define MSG_OK (msg_t)0

  OK wakeup message.
- #define MSG_TIMEOUT (msg_t)-1

  Wake-up caused by a timeout condition.
- #define MSG_RESET (msg_t)-2

  Wake-up caused by a reset condition.

Special time constants

- #define TIME_IMMEDIATE ((sysinterval_t)-1)

  Zero time specification for some functions with a timeout specification.
- #define TIME_INFINITE ((sysinterval_t)0)

  Infinite time specification for all functions with a timeout specification.
- #define TIME_MAX_INTERVAL ((sysinterval_t)-2)

  Maximum interval constant usable as timeout.
- #define TIME_MAX_SYSTIME ((systime_t)-1)

  Maximum system of system time before it wraps.

Thread state related macros

- #define NIL_STATE_WTSTART (tstate_t)0

  Thread not yet started or terminated.
- #define NIL_STATE_READY (tstate_t)1

  Thread ready or executing.
- #define NIL_STATE_SLEEPING (tstate_t)2

  Thread sleeping.
- #define NIL_STATE_SUSPENDED (tstate_t)3

  Thread suspended.
- #define NIL_STATE_WTEXIT (tstate_t)4

  Waiting a thread.
- #define NIL_STATE_WTQUEUE (tstate_t)5

  On queue or semaph.
- #define NIL_STATE_WTOREVT (tstate_t)6

  Waiting for events.
- #define NIL_STATE_WTANDEVT (tstate_t)7

  Waiting for events.
- #define NIL_STATE_SNDMSGQ (tstate_t)8

  Sending a message, in queue.
- #define NIL_STATE_WTMSG (tstate_t)1

  Waiting for a message.
- #define NIL_STATE_FINAL (tstate_t)1

  Thread terminated.
- #define NIL_THD_IS_WTSTART(tp)
- #define NIL_THD_IS_READY(tp)
- #define NIL_THD_IS_SLEEPING(tp)
- #define NIL_THD_IS_SUSPENDED(tp)
- #define NIL_THD_IS_WTEXIT(tp)
- #define NIL_THD_IS_WTQUEUE(tp)
- #define NIL_THD_IS_WTOREVT(tp)
- #define NIL_THD_IS_WTANDEVT(tp)
- #define NIL_THD_IS_SNDMSGQ(tp)
- #define NIL_THD_IS_WTMSG(tp)
- #define NIL_THD_IS_FINAL(tp)
- #define CH_STATE_NAMES

RT options not existing in NIL

- #define CH_CFG_USE_REGISTRY FALSE

Threads tables definition macros

- #define THD_TABLE_BEGIN const thread_descriptor_t nil_thd_configs[ ] = {

  Start of user threads table.
- #define THD_TABLE_THREAD(_prio, _name, _wap, _funcp, _arg)

  Entry of user threads table.
- #define THD_TABLE_END

  End of user threads table.

Memory alignment support macros

- #define MEM_ALIGN_MASK(a)

  Alignment mask constant.
- #define MEM_ALIGN_PREV(p, a)

  Aligns to the previous aligned memory address.
- #define MEM_ALIGN_NEXT(p, a)

  Aligns to the new aligned memory address.
- #define MEM_IS_ALIGNED(p, a)

  Returns whatever a pointer or memory size is aligned.
- #define MEM_IS_VALID_ALIGNMENT(a)

  Returns whatever a constant is a valid alignment.

Working Areas

- #define THD_WORKING_AREA_SIZE(n)

  Calculates the total Working Area size.
- #define THD_WORKING_AREA(s, n)

  Static working area allocation.

Threads abstraction macros

- #define THD_FUNCTION(tname, arg)

  Thread declaration macro.

ISRs abstraction macros

- #define CH_IRQ_IS_VALID_PRIORITY(prio)

  Priority level validation macro.
- #define CH_IRQ_IS_VALID_KERNEL_PRIORITY(prio)

  Priority level validation macro.
- #define CH_IRQ_PROLOGUE()

  IRQ handler enter code.
- #define CH_IRQ_EPILOGUE()

  IRQ handler exit code.
- #define CH_IRQ_HANDLER(id)

  Standard normal IRQ handler declaration.

Fast ISRs abstraction macros

- #define CH_FAST_IRQ_HANDLER(id)

  Standard fast IRQ handler declaration.

Time conversion utilities

- #define TIME_S2I(secs)

  Seconds to time interval.
- #define TIME_MS2I(msecs)

  Milliseconds to time interval.
- #define TIME_US2I(usecs)

  Microseconds to time interval.
- #define TIME_I2S(interval)

  Time interval to seconds.
- #define TIME_I2MS(interval)

  Time interval to milliseconds.
- #define TIME_I2US(interval)

  Time interval to microseconds.

Threads queues

- #define __THREADS_QUEUE_DATA(name)

  Data part of a static threads queue object initializer.
- #define THREADS_QUEUE_DECL(name)

  Static threads queue object initializer.

Macro Functions

- #define chSysGetRealtimeCounterX()

  Returns the current value of the system real time counter.
- #define chSysDisable()

  Raises the system interrupt priority mask to the maximum level.
- #define chSysSuspend()

  Raises the system interrupt priority mask to system level.
- #define chSysEnable()

  Lowers the system interrupt priority mask to user level.
- #define chSysLock()

  Enters the kernel lock state.
- #define chSysUnlock()

  Leaves the kernel lock state.
- #define chSysLockFromISR()

  Enters the kernel lock state from within an interrupt handler.
- #define chSysUnlockFromISR()

  Leaves the kernel lock state from within an interrupt handler.
- #define chSchGoSleepS(newstate)

  Puts the current thread to sleep into the specified state.
- #define chSchWakeupS(ntp, msg)

  Wakes up a thread.

- #define chSchIsRescRequiredI()

  Evaluates if a reschedule is required.
- #define chThdGetSelfX()

  Returns a pointer to the current `thread_t`.
- #define chThdGetPriorityX(void)

  Returns the current thread priority.
- #define chThdResumeS(trp, msg)

  Wakes up a thread waiting on a thread reference object.
- #define chThdSleepSeconds(secs)

  Delays the invoking thread for the specified number of seconds.
- #define chThdSleepMilliseconds(msecs)

  Delays the invoking thread for the specified number of milliseconds.
- #define chThdSleepMicroseconds(usecs)

  Delays the invoking thread for the specified number of microseconds.
- #define chThdSleepS(timeout)

  Suspends the invoking thread for the specified time.
- #define chThdSleepUntilS(abstime)

  Suspends the invoking thread until the system time arrives to the specified value.
- #define chThdQueueObjectInit(tqp)

  Initializes a threads queue object.
- #define chThdQueueIsEmptyI(tqp)

  Evaluates to `true` *if the specified queue is empty.*
- #define chVTGetSystemTimeX()

  Current system time.
- #define chVTTimeElapsedSinceX(start)

  Returns the elapsed time since the specified start time.
- #define chVTIsSystemTimeWithinX(start, end)

  Checks if the current system time is within the specified time window.
- #define chTimeAddX(systime, interval)

  Adds an interval to a system time returning a system time.
- #define chTimeDiffX(start, end)

  Subtracts two system times returning an interval.
- #define chDbgCheck(c)

  Function parameters check.
- #define chDbgAssert(c, r)

  Condition assertion.

Data Structures

- struct nil_threads_queue

  Structure representing a queue of threads.
- struct nil_thread_descriptor

  Structure representing a thread descriptor.
- struct nil_thread

  Structure representing a thread.
- struct nil_os_instance

  System data structure.

Macros

- #define __dbg_check_disable()
- #define __dbg_check_suspend()
- #define __dbg_check_enable()
- #define __dbg_check_lock()
- #define __dbg_check_unlock()
- #define __dbg_check_lock_from_isr()
- #define __dbg_check_unlock_from_isr()
- #define __dbg_check_enter_isr()
- #define __dbg_check_leave_isr()
- #define chDbgCheckClassI()
- #define chDbgCheckClassS()
- #define __CHIBIOS_NIL__

  ChibiOS/NIL identification macro.
- #define CH_KERNEL_STABLE 1

  Stable release flag.
- #define CH_CFG_ST_TIMEDELTA 0
- #define CH_CFG_USE_MESSAGES FALSE
- #define NIL_DBG_ENABLED TRUE
- #define THD_IDLE_BASE (&__main_thread_stack_base__)
- #define THD_IDLE_END (&__main_thread_stack_end__)
- #define CH_PORT_SUPPORTS_RECURSIVE_LOCKS FALSE
- #define __dbg_enter_lock()
- #define __dbg_enter_lock()
- #define __dbg_leave_lock()
- #define __dbg_leave_lock()
- #define __CH_STRINGIFY(a)

  Utility to make the parameter a quoted string.
- #define THD_WORKING_AREA_END(wa)

  Returns the top address of a working area.

Typedefs

- typedef uint32_t systime_t

  Type of system time.
- typedef uint32_t sysinterval_t

  Type of time interval.
- typedef uint64_t time_conv_t

  Type of time conversion variable.
- typedef struct nil_os_instance os_instance_t

  Type of a structure representing the system.
- typedef void(* tfunc_t) (void *p)

  Thread function.
- typedef struct nil_thread_descriptor thread_descriptor_t

  Type of a thread descriptor.
- typedef struct nil_thread thread_t

  Type of a structure representing a thread.
- typedef thread_t * thread_reference_t

  Type of a thread reference.
- typedef struct nil_threads_queue threads_queue_t

  Type of a queue of threads.
- typedef threads_queue_t semaphore_t

  Type of a structure representing a semaphore.

Functions

- thread_t * nil_find_thread (tstate_t state, void *p)

  Retrieves the highest priority thread in the specified state and associated to the specified object.
- cnt_t nil_ready_all (void *p, cnt_t cnt, msg_t msg)

  Puts in ready state all thread matching the specified status and associated object.
- void __dbg_check_disable (void)

  Guard code for `chSysDisable()`.
- void __dbg_check_suspend (void)

  Guard code for `chSysSuspend()`.
- void __dbg_check_enable (void)

  Guard code for `chSysEnable()`.
- void __dbg_check_lock (void)

  Guard code for `chSysLock()`.
- void __dbg_check_unlock (void)

  Guard code for `chSysUnlock()`.
- void __dbg_check_lock_from_isr (void)

  Guard code for `chSysLockFromIsr()`.
- void __dbg_check_unlock_from_isr (void)

  Guard code for `chSysUnlockFromIsr()`.
- void __dbg_check_enter_isr (void)

  Guard code for `CH_IRQ_PROLOGUE()`.
- void __dbg_check_leave_isr (void)

  Guard code for `CH_IRQ_EPILOGUE()`.
- void chDbgCheckClassI (void)

  I-class functions context check.
- void chDbgCheckClassS (void)

  S-class functions context check.
- void chSysInit (void)

  Initializes the kernel.
- void chSysHalt (const char *reason)

  Halts the system.
- void chSysTimerHandlerI (void)

  Time management handler.
- void chSysUnconditionalLock (void)

  Unconditionally enters the kernel lock state.
- void chSysUnconditionalUnlock (void)

  Unconditionally leaves the kernel lock state.
- syssts_t chSysGetStatusAndLockX (void)

  Returns the execution status and enters a critical zone.
- void chSysRestoreStatusX (syssts_t sts)

  Restores the specified execution status and leaves a critical zone.
- bool chSysIsCounterWithinX (rtcnt_t cnt, rtcnt_t start, rtcnt_t end)

  Realtime window test.
- void chSysPolledDelayX (rtcnt_t cycles)

  Polled delay.
- thread_t * chSchReadyI (thread_t *tp, msg_t msg)

  Makes the specified thread ready for execution.
- bool chSchIsPreemptionRequired (void)

  Evaluates if preemption is required.
- void chSchDoPreemption (void)

Switches to the first thread on the runnable queue.

- void chSchRescheduleS (void)

  Reschedules if needed.
- msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)

  Puts the current thread to sleep into the specified state with timeout specification.
- bool chTimeIsInRangeX (systime_t time, systime_t start, systime_t end)

  Checks if the specified time is within the specified time range.
- thread_t * chThdCreateI (const thread_descriptor_t *tdp)

  Creates a new thread into a static memory area.
- thread_t * chThdCreate (const thread_descriptor_t *tdp)

  Creates a new thread into a static memory area.
- void chThdExit (msg_t msg)

  Terminates the current thread.
- msg_t chThdWait (thread_t *tp)

  Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
- msg_t chThdSuspendTimeoutS (thread_reference_t *trp, sysinterval_t timeout)

  Sends the current thread sleeping and sets a reference variable.
- void chThdResumeI (thread_reference_t *trp, msg_t msg)

  Wakes up a thread waiting on a thread reference object.
- void chThdResume (thread_reference_t *trp, msg_t msg)

  Wakes up a thread waiting on a thread reference object.
- void chThdSleep (sysinterval_t timeout)

  Suspends the invoking thread for the specified time.
- void chThdSleepUntil (systime_t abstime)

  Suspends the invoking thread until the system time arrives to the specified value.
- msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)

  Enqueues the caller thread on a threads queue object.
- void chThdDoDequeueNextI (threads_queue_t *tqp, msg_t msg)

  Dequeues and wakes up one thread from the threads queue object.
- void chThdDequeueNextI (threads_queue_t *tqp, msg_t msg)

  Dequeues and wakes up one thread from the threads queue object, if any.
- void chThdDequeueAllI (threads_queue_t *tqp, msg_t msg)

  Dequeues and wakes up all threads from the threads queue object.

Variables

- os_instance_t nil

  System data structures.

6.2.3.2 Macro Definition Documentation

6.2.3.2.1 __dbg_check_disable

```
void __dbg_check_disable()
```

6.2.3.2.2 __dbg_check_suspend

```
void __dbg_check_suspend()
```

6.2.3.2.3 __dbg_check_enable

```
void __dbg_check_enable()
```

6.2.3.2.4 __dbg_check_lock

```
void __dbg_check_lock()
```

6.2.3.2.5 __dbg_check_unlock

```
void __dbg_check_unlock()
```

6.2.3.2.6 __dbg_check_lock_from_isr

```
void __dbg_check_lock_from_isr()
```

6.2.3.2.7 __dbg_check_unlock_from_isr

```
void __dbg_check_unlock_from_isr()
```

6.2.3.2.8 __dbg_check_enter_isr

```
void __dbg_check_enter_isr()
```

6.2.3.2.9 __dbg_check_leave_isr

```
void __dbg_check_leave_isr()
```

6.2.3.2.10 chDbgCheckClassI

```
void chDbgCheckClassI()
```

6.2.3.2.11 chDbgCheckClassS

```
void chDbgCheckClassS()
```

6.2.3.2.12 __CHIBIOS_NIL__

```
#define __CHIBIOS_NIL__
```

ChibiOS/NIL identification macro.

6.2.3.2.13 CH_KERNEL_STABLE

```
#define CH_KERNEL_STABLE 1
```

Stable release flag.

6.2.3.2.14 CH_KERNEL_VERSION

```
#define CH_KERNEL_VERSION "4.1.3"
```

Kernel version string.

6.2.3.2.15 CH_KERNEL_MAJOR

```
#define CH_KERNEL_MAJOR 4
```

Kernel version major number.

6.2.3.2.16 CH_KERNEL_MINOR

```
#define CH_KERNEL_MINOR 1
```

Kernel version minor number.

6.2.3.2.17 CH_KERNEL_PATCH

```
#define CH_KERNEL_PATCH 3
```

Kernel version patch number.

6.2.3.2.18 FALSE

```
#define FALSE 0
```

Generic 'false' preprocessor boolean constant.

Note

It is meant to be used in configuration files as switch.

6.2.3.2.19 TRUE

```
#define TRUE 1
```

Generic 'true' preprocessor boolean constant.

Note

It is meant to be used in configuration files as switch.

6.2.3.2.20 MSG_OK

```
#define MSG_OK (msg_t)0
```

OK wakeup message.

6.2.3.2.21 MSG_TIMEOUT

```
#define MSG_TIMEOUT (msg_t)-1
```

Wake-up caused by a timeout condition.

6.2.3.2.22 MSG_RESET

```
#define MSG_RESET (msg_t)-2
```

Wake-up caused by a reset condition.

6.2.3.2.23 TIME_IMMEDIATE

```
#define TIME_IMMEDIATE ((sysinterval_t)-1)
```

Zero time specification for some functions with a timeout specification.

Note

Not all functions accept `TIME_IMMEDIATE` as timeout parameter, see the specific function documentation.

6.2.3.2.24 TIME_INFINITE

```
#define TIME_INFINITE ((sysinterval_t)0)
```

Infinite time specification for all functions with a timeout specification.

6.2.3.2.25 TIME_MAX_INTERVAL

```
#define TIME_MAX_INTERVAL ((sysinterval_t)-2)
```

Maximum interval constant usable as timeout.

6.2.3.2.26 TIME_MAX_SYSTIME

```
#define TIME_MAX_SYSTIME ((systime_t)-1)
```

Maximum system of system time before it wraps.

6.2.3.2.27 NIL_STATE_WTSTART

```
#define NIL_STATE_WTSTART (tstate_t)0
```

Thread not yet started or terminated.

6.2.3.2.28 NIL_STATE_READY

```
#define NIL_STATE_READY (tstate_t)1
```

Thread ready or executing.

6.2.3.2.29 NIL_STATE_SLEEPING

```
#define NIL_STATE_SLEEPING (tstate_t)2
```

Thread sleeping.

6.2.3.2.30 NIL_STATE_SUSPENDED

```
#define NIL_STATE_SUSPENDED (tstate_t)3
```

Thread suspended.

6.2.3.2.31 NIL_STATE_WTEXIT

```
#define NIL_STATE_WTEXIT (tstate_t)4
```

Waiting a thread.

6.2.3.2.32 NIL_STATE_WTQUEUE

```
#define NIL_STATE_WTQUEUE (tstate_t)5
```

On queue or semaph.

6.2.3.2.33 NIL_STATE_WTOREVT

```
#define NIL_STATE_WTOREVT (tstate_t)6
```

Waiting for events.

6.2.3.2.34 NIL_STATE_WTANDEVT

```
#define NIL_STATE_WTANDEVT (tstate_t)7
```

Waiting for events.

6.2.3.2.35 NIL_STATE_SNDMSGQ

```
#define NIL_STATE_SNDMSGQ (tstate_t)8
```

Sending a message, in queue.

6.2.3.2.36 NIL_STATE_WTMSG

```
#define NIL_STATE_WTMSG (tstate_t)1
```

Waiting for a message.

6.2.3.2.37 NIL_STATE_FINAL

```
#define NIL_STATE_FINAL (tstate_t)1
```

Thread terminated.

6.2.3.2.38 NIL_THD_IS_WTSTART

```
#define NIL_THD_IS_WTSTART(
              tp)
```

Value:

```
((tp)->state == NIL_STATE_WTSTART)
```

6.2.3.2.39 NIL_THD_IS_READY

```
#define NIL_THD_IS_READY(
              tp)
```

Value:

```
((tp)->state == NIL_STATE_READY)
```

6.2.3.2.40 NIL_THD_IS_SLEEPING

```
#define NIL_THD_IS_SLEEPING(
              tp)
```

Value:

```
((tp)->state == NIL_STATE_SLEEPING)
```

6.2.3.2.41 NIL_THD_IS_SUSPENDED

```
#define NIL_THD_IS_SUSPENDED(
              tp)
```

Value:

```
((tp)->state == NIL_STATE_SUSPENDED)
```

6.2.3.2.42 NIL_THD_IS_WTEXIT

```
#define NIL_THD_IS_WTEXIT(
              tp)
```

Value:

```
((tp)->state == NIL_STATE_WTEXIT)
```

6.2.3.2.43 NIL_THD_IS_WTQUEUE

```
#define NIL_THD_IS_WTQUEUE(
              tp)
```

Value:

```
((tp)->state == NIL_STATE_WTQUEUE)
```

6.2.3.2.44 NIL_THD_IS_WTOREVT

```
#define NIL_THD_IS_WTOREVT(
              tp)
```

Value:

```
((tp)->state == NIL_STATE_WTOREVT)
```

6.2.3.2.45 NIL_THD_IS_WTANDEVT

```
#define NIL_THD_IS_WTANDEVT(
              tp)
```

Value:

```
((tp)->state == NIL_STATE_WTANDEVT)
```

6.2.3.2.46 NIL_THD_IS_SNDMSGQ

```
#define NIL_THD_IS_SNDMSGQ(
              tp)
```

Value:

```
((tp)->state == NIL_STATE_SNDMSGQ)
```

6.2.3.2.47 NIL_THD_IS_WTMSG

```
#define NIL_THD_IS_WTMSG(
              tp)
```

Value:

```
((tp)->state == NIL_STATE_WTMSG)
```

6.2.3.2.48 NIL_THD_IS_FINAL

```
#define NIL_THD_IS_FINAL(
              tp)
```

Value:

```
((tp)->state == NIL_STATE_FINAL)
```

6.2.3.2.49 CH_STATE_NAMES

```
#define CH_STATE_NAMES
```

Value:

```
  "WTSTART", "READY", "SLEEPING", "SUSPENDED", "WTEXIT", "WTQUEUE",         \
  "WTOREVT", "WTANDEVT", "SNDMSGQ", "SNDMSG", "WTMSG", "FINAL"
```

6.2.3.2.50 CH_CFG_USE_REGISTRY

```
#define CH_CFG_USE_REGISTRY FALSE
```

6.2.3.2.51 CH_CFG_ST_TIMEDELTA

```
#define CH_CFG_ST_TIMEDELTA 0
```

6.2.3.2.52 CH_CFG_USE_MESSAGES

```
#define CH_CFG_USE_MESSAGES FALSE
```

6.2.3.2.53 NIL_DBG_ENABLED

```
#define NIL_DBG_ENABLED TRUE
```

6.2.3.2.54 THD_IDLE_BASE

```
#define THD_IDLE_BASE (&__main_thread_stack_base__)
```

Boundaries of the idle thread boundaries, only required if stack checking is enabled.

6.2.3.2.55 THD_IDLE_END

```
#define THD_IDLE_END (&__main_thread_stack_end__)
```

6.2.3.2.56 CH_PORT_SUPPORTS_RECURSIVE_LOCKS

```
#define CH_PORT_SUPPORTS_RECURSIVE_LOCKS FALSE
```

6.2.3.2.57 __dbg_enter_lock [1/2]

```
#define __dbg_enter_lock()
```

Value:

```
(nil.lock_cnt = (cnt_t)1)
```

6.2.3.2.58 __dbg_enter_lock [2/2]

```
#define __dbg_enter_lock()
```

6.2.3.2.59 __dbg_leave_lock [1/2]

```
#define __dbg_leave_lock()
```

Value:

```
(nil.lock_cnt = (cnt_t)0)
```

6.2.3.2.60 __dbg_leave_lock [2/2]

```
#define __dbg_leave_lock()
```

6.2.3.2.61 __CH_STRINGIFY

```
#define __CH_STRINGIFY(
              a)
```

Value:

```
#a
```

Utility to make the parameter a quoted string.

6.2.3.2.62 THD_TABLE_BEGIN

```
#define THD_TABLE_BEGIN  const thread_descriptor_t nil_thd_configs[] = {
```

Start of user threads table.

6.2.3.2.63 THD_TABLE_THREAD

```
#define THD_TABLE_THREAD(
              _prio,
              _name,
              _wap,
              _funcp,
              _arg)
```

Value:

```
  {                                                                     \
    .name  = (_name),                                                   \
    .wbase = (_wap),                                                    \
    .wend  = THD_WORKING_AREA_END(_wap),                                \
    .prio  = (_prio),                                                   \
    .funcp = (_funcp),                                                  \
    .arg   = (_arg)                                                     \
  },
```

Entry of user threads table.

6.2.3.2.64 THD_TABLE_END

```
#define THD_TABLE_END
```

Value:

```
  {                                                                     \
    .name  = "idle",                                                    \
    .wbase = THD_IDLE_BASE,                                             \
    .wend  = THD_IDLE_END,                                              \
    .prio  = CH_CFG_MAX_THREADS,                                        \
    .funcp = NULL,                                                      \
    .arg   = NULL                                                       \
  }                                                                     \
};
```

End of user threads table.

6.2.3.2.65 MEM_ALIGN_MASK

```
#define MEM_ALIGN_MASK(
              a)
```

Value:

```
((size_t)(a) - 1U)
```

Alignment mask constant.

Parameters

in	*a*	alignment, must be a power of two

6.2.3.2.66 MEM_ALIGN_PREV

```
#define MEM_ALIGN_PREV(
              p,
              a)
```

Value:

```
((size_t)(p) & ~MEM_ALIGN_MASK(a))
```

Aligns to the previous aligned memory address.

Parameters

in	*p*	variable to be aligned
in	*a*	alignment, must be a power of two

6.2.3.2.67 MEM_ALIGN_NEXT

```
#define MEM_ALIGN_NEXT(
              p,
              a)
```

Value:

```
                                        MEM_ALIGN_PREV((size_t)(p) +                \
MEM_ALIGN_MASK(a), (a))
```

Aligns to the new aligned memory address.

Parameters

in	*p*	variable to be aligned
in	*a*	alignment, must be a power of two

6.2.3.2.68 MEM_IS_ALIGNED

```
#define MEM_IS_ALIGNED(
              p,
              a)
```

Value:

```
(((size_t)(p) & MEM_ALIGN_MASK(a)) == 0U)
```

Returns whatever a pointer or memory size is aligned.

Parameters

in	*p*	variable to be aligned
in	*a*	alignment, must be a power of two

6.2.3.2.69 MEM_IS_VALID_ALIGNMENT

```
#define MEM_IS_VALID_ALIGNMENT(
              a)
```

Value:

```
  (((size_t)(a) != 0U) && (((size_t)(a) & ((size_t)(a) - 1U)) == 0U))
```

Returns whatever a constant is a valid alignment.

Valid alignments are powers of two.

Parameters

in	*a*	alignment to be checked, must be a constant

6.2.3.2.70 THD_WORKING_AREA_SIZE

```
#define THD_WORKING_AREA_SIZE(
              n)
```

Value:

```
                                        MEM_ALIGN_NEXT(PORT_WA_SIZE(n),             \
PORT_STACK_ALIGN)
```

Calculates the total Working Area size.

Parameters

in	*n*	the stack size to be assigned to the thread

Returns

The total used memory in bytes.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.2.71 THD_WORKING_AREA

```
#define THD_WORKING_AREA(
              s,
              n)
```

Value:

```
PORT_WORKING_AREA(s, n)
```

Static working area allocation.

This macro is used to allocate a static thread working area aligned as both position and size.

Parameters

in	*s*	the name to be assigned to the stack array
in	*n*	the stack size to be assigned to the thread

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.2.72 THD_WORKING_AREA_END

```
#define THD_WORKING_AREA_END(
              wa)
```

Value:

```
  ((wa) + ((sizeof wa) / sizeof (stkalign_t)))
```

Returns the top address of a working area.

Note

The parameter is assumed to be an array of `stkalign_t`. The macros is invalid for anything else.

Parameters

in	*wa*	working area array

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.2.73 THD_FUNCTION

```
#define THD_FUNCTION(
              tname,
              arg)
```

Value:

```
PORT_THD_FUNCTION(tname, arg)
```

Thread declaration macro.

Note

Thread declarations should be performed using this macro because the port layer could define optimizations for thread functions.

6.2.3.2.74 CH_IRQ_IS_VALID_PRIORITY

```
#define CH_IRQ_IS_VALID_PRIORITY(
              prio)
```

Value:

```
  PORT_IRQ_IS_VALID_PRIORITY(prio)
```

Priority level validation macro.

This macro determines if the passed value is a valid priority level for the underlying architecture.

Parameters

in	*prio*	the priority level

Returns

Priority range result.

Return values

false	if the priority is invalid or if the architecture does not support priorities.
true	if the priority is valid.

6.2.3.2.75 CH_IRQ_IS_VALID_KERNEL_PRIORITY

```
#define CH_IRQ_IS_VALID_KERNEL_PRIORITY(
              prio)
```

Value:

```
  PORT_IRQ_IS_VALID_KERNEL_PRIORITY(prio)
```

Priority level validation macro.

This macro determines if the passed value is a valid priority level that cannot preempt the kernel critical zone.

Parameters

in	*prio*	the priority level

Returns

Priority range result.

Return values

false	if the priority is invalid or if the architecture does not support priorities.
true	if the priority is valid.

6.2.3.2.76 CH_IRQ_PROLOGUE

```
#define CH_IRQ_PROLOGUE()
```

Value:

```
PORT_IRQ_PROLOGUE();                                                    \
__dbg_check_enter_isr()
```

IRQ handler enter code.

Note

Usually IRQ handlers functions are also declared naked.

On some architectures this macro can be empty.

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.2.77 CH_IRQ_EPILOGUE

```
#define CH_IRQ_EPILOGUE()
```

Value:

```
__dbg_check_leave_isr();                                                \
PORT_IRQ_EPILOGUE()
```

IRQ handler exit code.

Note

Usually IRQ handlers function are also declared naked.

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.2.78 CH_IRQ_HANDLER

```
#define CH_IRQ_HANDLER(
              id)
```

Value:

```
PORT_IRQ_HANDLER(id)
```

Standard normal IRQ handler declaration.

Note

`id` can be a function name or a vector number depending on the port implementation.

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.2.79 CH_FAST_IRQ_HANDLER

```
#define CH_FAST_IRQ_HANDLER(
              id)
```

Value:

```
PORT_FAST_IRQ_HANDLER(id)
```

Standard fast IRQ handler declaration.

Note

`id` can be a function name or a vector number depending on the port implementation.

Not all architectures support fast interrupts.

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.2.80 TIME_S2I

```
#define TIME_S2I(
              secs)
```

Value:

```
  ((sysinterval_t)((time_conv_t)(secs) * (time_conv_t)CH_CFG_ST_FREQUENCY))
```

Seconds to time interval.

Converts from seconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	*secs*	number of seconds

Returns

The number of ticks.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.2.81 TIME_MS2I

```
#define TIME_MS2I(
              msecs)
```

Value:

```
  ((sysinterval_t)((((time_conv_t)(msecs) *                                \
                    (time_conv_t)CH_CFG_ST_FREQUENCY) +                    \
                   (time_conv_t)999) / (time_conv_t)1000))
```

Milliseconds to time interval.

Converts from milliseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	*msecs*	number of milliseconds

Returns

The number of ticks.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.2.82 TIME_US2I

```
#define TIME_US2I(
              usecs)
```

Value:

```
  ((sysinterval_t)((((time_conv_t)(usecs) *                                \
                    (time_conv_t)CH_CFG_ST_FREQUENCY) +                    \
                   (time_conv_t)999999) / (time_conv_t)1000000))
```

Microseconds to time interval.

Converts from microseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	*usecs*	number of microseconds

Returns

The number of ticks.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.2.83 TIME_I2S

```
#define TIME_I2S(
                interval)
```

Value:

```
  (time_secs_t)(((time_conv_t)(interval) +                                  \
                 (time_conv_t)CH_CFG_ST_FREQUENCY -                         \
                 (time_conv_t)1) / (time_conv_t)CH_CFG_ST_FREQUENCY)
```

Time interval to seconds.

Converts from system ticks number to seconds.

Note

The result is rounded up to the next second boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	*interval*	interval in ticks

Returns

The number of seconds.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.2.84 TIME_I2MS

```
#define TIME_I2MS(
                interval)
```

Value:

```
  (time_msecs_t)((((time_conv_t)(interval) * (time_conv_t)1000) +           \
                  (time_conv_t)CH_CFG_ST_FREQUENCY - (time_conv_t)1) /       \
                 (time_conv_t)CH_CFG_ST_FREQUENCY)
```

Time interval to milliseconds.

Converts from system ticks number to milliseconds.

Note

The result is rounded up to the next millisecond boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	*interval*	interval in ticks

Returns

The number of milliseconds.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.2.85 TIME_I2US

```
#define TIME_I2US(
              interval)
```

Value:

```
    (time_usecs_t)((((time_conv_t)(interval) * (time_conv_t)1000000) +      \
                    (time_conv_t)CH_CFG_ST_FREQUENCY - (time_conv_t)1) /    \
                   (time_conv_t)CH_CFG_ST_FREQUENCY)
```

Time interval to microseconds.

Converts from system ticks number to microseconds.

Note

The result is rounded up to the next microsecond boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	*interval*	interval in ticks

Returns

The number of microseconds.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.2.86 __THREADS_QUEUE_DATA

```
#define __THREADS_QUEUE_DATA(
              name)
```

Value:

```
{(cnt_t)0}
```

Data part of a static threads queue object initializer.

This macro should be used when statically initializing a threads queue that is part of a bigger structure.

Parameters

in	*name*	the name of the threads queue variable

6.2.3.2.87 THREADS_QUEUE_DECL

```
#define THREADS_QUEUE_DECL(
              name)
```

Value:

```
threads_queue_t name = __THREADS_QUEUE_DATA(name)
```

Static threads queue object initializer.

Statically initialized threads queues require no explicit initialization using `queue_init()`.

Parameters

in	*name*	the name of the threads queue variable

6.2.3.2.88 chSysGetRealtimeCounterX

```
#define chSysGetRealtimeCounterX()
```

Value:

```
(rtcnt_t)port_rt_get_counter_value()
```

Returns the current value of the system real time counter.

Note

This function is only available if the port layer supports the option `PORT_SUPPORTS_RT`.

Returns

The value of the system realtime counter of type rtcnt_t.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.2.3.2.89 chSysDisable

```
#define chSysDisable()
```

Value:

```
  {                                                   \
  port_disable();                                     \
  __dbg_check_disable();                              \
}
```

Raises the system interrupt priority mask to the maximum level.

All the maskable interrupt sources are disabled regardless their hardware priority.

Note

Do not invoke this API from within a kernel lock.

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.2.90 chSysSuspend

```
#define chSysSuspend()
```

Value:

```
  {                                                   \
  port_suspend();                                     \
  __dbg_check_suspend();                              \
}
```

Raises the system interrupt priority mask to system level.

The interrupt sources that should not be able to preempt the kernel are disabled, interrupt sources with higher priority are still enabled.

Note

Do not invoke this API from within a kernel lock.

This API is no replacement for `chSysLock()`, the `chSysLock()` could do more than just disable the interrupts.

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.2.91 chSysEnable

```
#define chSysEnable()
```

Value:

```
  {                                                   \
  __dbg_check_enable();                                \
  port_enable();                                       \
}
```

Lowers the system interrupt priority mask to user level.

All the interrupt sources are enabled.

Note

Do not invoke this API from within a kernel lock.

This API is no replacement for `chSysUnlock()`, the `chSysUnlock()` could do more than just enable the interrupts.

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.2.92 chSysLock

```
#define chSysLock()
```

Value:

```
  {                                                    \
  port_lock();                                         \
  __dbg_check_lock();                                  \
}
```

Enters the kernel lock state.

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.2.93 chSysUnlock

```
#define chSysUnlock()
```

Value:

```
  {                                                   \
  __dbg_check_unlock();                                \
  port_unlock();                                       \
}
```

Leaves the kernel lock state.

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.2.94 chSysLockFromISR

```
#define chSysLockFromISR()
```

Value:

```
  {                                                 \
  port_lock_from_isr();                                                 \
  __dbg_check_lock_from_isr();                                          \
}
```

Enters the kernel lock state from within an interrupt handler.

Note

This API may do nothing on some architectures, it is required because on ports that support preemptable interrupt handlers it is required to raise the interrupt mask to the same level of the system mutual exclusion zone.
It is good practice to invoke this API before invoking any I-class syscall from an interrupt handler.

This API must be invoked exclusively from interrupt handlers.

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.2.95 chSysUnlockFromISR

```
#define chSysUnlockFromISR()
```

Value:

```
  {                                                 \
  __dbg_check_unlock_from_isr();                                        \
  port_unlock_from_isr();                                               \
}
```

Leaves the kernel lock state from within an interrupt handler.

Note

This API may do nothing on some architectures, it is required because on ports that support preemptable interrupt handlers it is required to raise the interrupt mask to the same level of the system mutual exclusion zone.
It is good practice to invoke this API after invoking any I-class syscall from an interrupt handler.

This API must be invoked exclusively from interrupt handlers.

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.2.96 chSchGoSleepS

```
#define chSchGoSleepS(
              newstate)
```

Value:

```
chSchGoSleepTimeoutS(newstate, TIME_INFINITE)
```

Puts the current thread to sleep into the specified state.

Parameters

in	*newstate*	the new thread state or a semaphore pointer

Returns

The wakeup message.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

6.2.3.2.97 chSchWakeupS

```
#define chSchWakeupS(
              ntp,
              msg)
```

Value:

```
  do {                                      \
  chSchReadyI(ntp, msg);                                  \
  chSchRescheduleS();                                     \
} while (false)
```

Wakes up a thread.

Parameters

in	*ntp*	the thread to be made ready
in	*msg*	the wakeup message

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

6.2.3.2.98 chSchIsRescRequiredI

```
#define chSchIsRescRequiredI()
```

Value:

```
((bool)(nil.current != nil.next))
```

Evaluates if a reschedule is required.

Return values

true	if there is a thread that must go in running state immediately.
false	if preemption is not required.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.2.3.2.99 chThdGetSelfX

```
#define chThdGetSelfX()
```

Value:

```
nil.current
```

Returns a pointer to the current thread_t.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.2.3.2.100 chThdGetPriorityX

```
#define chThdGetPriorityX(
              void)
```

Value:

```
(tprio_t)(nil.current - &nil.threads[0])
```

Returns the current thread priority.

Note

Can be invoked in any context.

Returns

The current thread priority.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.2.3.2.101 chThdResumeS

```
#define chThdResumeS(
              trp,
              msg)
```

Value:

```
  do {                                      \
  chThdResumeI(trp, msg);                                   \
  chSchRescheduleS();                                       \
} while (false)
```

Wakes up a thread waiting on a thread reference object.

Note

This function must reschedule, it can only be called from thread context.

Parameters

in	*trp*	a pointer to a thread reference object
in	*msg*	the message code

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

6.2.3.2.102 chThdSleepSeconds

```
#define chThdSleepSeconds(
              secs)
```

Value:

```
chThdSleep(TIME_S2I(secs))
```

Delays the invoking thread for the specified number of seconds.

Note

The specified time is rounded up to a value allowed by the real system clock.

The maximum specified value is implementation dependent.

Parameters

in	*secs*	time in seconds, must be different from zero

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.2.103 chThdSleepMilliseconds

```
#define chThdSleepMilliseconds(
              msecs)
```

Value:

```
chThdSleep(TIME_MS2I(msecs))
```

Delays the invoking thread for the specified number of milliseconds.

Note

The specified time is rounded up to a value allowed by the real system clock.

The maximum specified value is implementation dependent.

Parameters

in	*msecs*	time in milliseconds, must be different from zero

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.2.104 chThdSleepMicroseconds

```
#define chThdSleepMicroseconds(
              usecs)
```

Value:

```
chThdSleep(TIME_US2I(usecs))
```

Delays the invoking thread for the specified number of microseconds.

Note

The specified time is rounded up to a value allowed by the real system clock.

The maximum specified value is implementation dependent.

Parameters

in	*usecs*	time in microseconds, must be different from zero

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.2.105 chThdSleepS

```
#define chThdSleepS(
              timeout)
```

Value:

```
  (void) chSchGoSleepTimeoutS(NIL_STATE_SLEEPING, timeout)
```

Suspends the invoking thread for the specified time.

Parameters

in	*timeout*	the delay in system ticks

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

6.2.3.2.106 chThdSleepUntilS

```
#define chThdSleepUntilS(
              abstime)
```

Value:

```
  (void) chSchGoSleepTimeoutS(NIL_STATE_SLEEPING,                          \
                              chTimeDiffX(chVTGetSystemTimeX(), (abstime)))
```

Suspends the invoking thread until the system time arrives to the specified value.

Parameters

in	*abstime*	absolute system time

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

6.2.3.2.107 chThdQueueObjectInit

```
#define chThdQueueObjectInit(
              tqp)
```

Value:

```
((tqp)->cnt = (cnt_t)0)
```

Initializes a threads queue object.

Parameters

out	*tqp*	pointer to the threads queue object

Function Class:

Object or module nitializer function.

6.2.3.2.108 chThdQueueIsEmptyI

```
#define chThdQueueIsEmptyI(
              tqp)
```

Value:

```
((bool)(tqp->cnt >= (cnt_t)0))
```

Evaluates to `true` if the specified queue is empty.

Parameters

out	*tqp*	pointer to the threads queue object

Returns

The queue status.

Return values

false	if the queue is not empty.
true	if the queue is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.2.3.2.109 chVTGetSystemTimeX

```
#define chVTGetSystemTimeX()
```

Value:

```
(nil.systime)
```

Current system time.

Returns the number of system ticks since the `chSysInit()` invocation.

Note

The counter can reach its maximum and then restart from zero.

This function can be called from any context but its atomicity is not guaranteed on architectures whose word size is less than `systime_t` size.

Returns

The system time in ticks.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.2.3.2.110 chVTTimeElapsedSinceX

```
#define chVTTimeElapsedSinceX(
              start)
```

Value:

```
  chTimeDiffX((start), chVTGetSystemTimeX())
```

Returns the elapsed time since the specified start time.

Parameters

in	*start*	start time

Returns

The elapsed time.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.2.3.2.111 chVTIsSystemTimeWithinX

```
#define chVTIsSystemTimeWithinX(
              start,
              end)
```

Value:

```
  chTimeIsInRangeX(chVTGetSystemTimeX(), start, end)
```

Checks if the current system time is within the specified time window.

Note

When start==end then the function returns always false because the time window has zero size.

Parameters

in	*start*	the start of the time window (inclusive)
in	*end*	the end of the time window (non inclusive)

Return values

true	current time within the specified time window.
false	current time not within the specified time window.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.2.3.2.112 chTimeAddX

```
#define chTimeAddX(
              systime,
              interval)
```

Value:

```
((systime_t)(systime) + (systime_t)(interval))
```

Adds an interval to a system time returning a system time.

Parameters

in	*systime*	base system time
in	*interval*	interval to be added

Returns

The new system time.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.2.3.2.113 chTimeDiffX

```
#define chTimeDiffX(
              start,
              end)
```

Value:

```
((sysinterval_t)((systime_t)((systime_t)(end) - (systime_t)(start))))
```

Subtracts two system times returning an interval.

Parameters

in	*start*	first system time
in	*end*	second system time

Returns

The interval representing the time difference.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.2.3.2.114 chDbgCheck

```
#define chDbgCheck(
              c)
```

Value:

```
  do {                                                      \
  /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/       \
  if (CH_DBG_ENABLE_CHECKS != FALSE) {                                      \
    if (!(c)) {                                                             \
  /*lint -restore*/                                                        \
      chSysHalt(__func__);                                                  \
    }                                                                       \
  }                                                                         \
} while (false)
```

Function parameters check.

If the condition check fails then the kernel panics and halts.

Note

The condition is tested only if the CH_DBG_ENABLE_CHECKS switch is specified in chconf.h else the macro does nothing.

Parameters

in	*c*	the condition to be verified to be true

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.2.115 chDbgAssert

```
#define chDbgAssert(
                c,
                r)
```

Value:

```
  do {                                                      \
  /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/       \
  if (CH_DBG_ENABLE_ASSERTS != FALSE) {                                     \
    if (!(c)) {                                                             \
  /*lint -restore*/                                                        \
      chSysHalt(__func__);                                                  \
    }                                                                       \
  }                                                                         \
} while (false)
```

Condition assertion.

If the condition check fails then the kernel panics with a message and halts.

Note

The condition is tested only if the CH_DBG_ENABLE_ASSERTS switch is specified in chconf.h else the macro does nothing.

The remark string is not currently used except for putting a comment in the code about the assertion.

Parameters

in	*c*	the condition to be verified to be true
in	*r*	a remark string

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.3 Typedef Documentation

6.2.3.3.1 rtcnt_t

```
typedef port_rtcnt_t rtcnt_t
```

Realtime counter.

6.2.3.3.2 syssts_t

```
typedef port_syssts_t syssts_t
```

System status word.

6.2.3.3.3 stkalign_t

```
typedef port_stkalign_t stkalign_t
```

Stack alignment type.

6.2.3.3.4 tstate_t

```
typedef uint8_t tstate_t
```

Thread state.

6.2.3.3.5 tprio_t

```
typedef uint32_t tprio_t
```

Thread priority.

6.2.3.3.6 msg_t

```
typedef int32_t msg_t
```

Inter-thread message.

6.2.3.3.7 eventid_t

```
typedef int32_t eventid_t
```

Numeric event identifier.

6.2.3.3.8 eventmask_t

```
typedef uint32_t eventmask_t
```

Mask of event identifiers.

6.2.3.3.9 eventflags_t

```
typedef uint32_t eventflags_t
```

Mask of event flags.

6.2.3.3.10 cnt_t

```
typedef int32_t cnt_t
```

Generic signed counter.

6.2.3.3.11 ucnt_t

```
typedef uint32_t ucnt_t
```

Generic unsigned counter.

6.2.3.3.12 systime_t

```
typedef uint32_t systime_t
```

Type of system time.

Note

It is selectable in configuration between 16 or 32 bits.

6.2.3.3.13 sysinterval_t

```
typedef uint32_t sysinterval_t
```

Type of time interval.

Note

It is selectable in configuration between 16 or 32 bits.

6.2.3.3.14 time_conv_t

```
typedef uint64_t time_conv_t
```

Type of time conversion variable.

Note

This type must have double width than other time types, it is only used internally for conversions.

6.2.3.3.15 os_instance_t

```
typedef struct nil_os_instance os_instance_t
```

Type of a structure representing the system.

6.2.3.3.16 tfunc_t

```
typedef void(* tfunc_t) (void *p)
```

Thread function.

6.2.3.3.17 thread_descriptor_t

```
typedef struct nil_thread_descriptor thread_descriptor_t
```

Type of a thread descriptor.

6.2.3.3.18 thread_t

```
typedef struct nil_thread thread_t
```

Type of a structure representing a thread.

Note

It is required as an early definition.

6.2.3.3.19 thread_reference_t

```
typedef thread_t* thread_reference_t
```

Type of a thread reference.

6.2.3.3.20 threads_queue_t

```
typedef struct nil_threads_queue threads_queue_t
```

Type of a queue of threads.

6.2.3.3.21 semaphore_t

```
typedef threads_queue_t semaphore_t
```

Type of a structure representing a semaphore.

Note

Semaphores are implemented on thread queues, the object is the same, the behavior is slightly different.

6.2.3.4 Function Documentation

6.2.3.4.1 nil_find_thread()

```
thread_t * nil_find_thread (
             tstate_t state,
             void * p)
```

Retrieves the highest priority thread in the specified state and associated to the specified object.

Parameters

in	*state*	thread state
in	*p*	object pointer

Returns

The pointer to the found thread.

Return values

NULL	if the thread is not found.

Function Class:

Not an API, this function is for internal use only.

6.2.3.4.2 nil_ready_all()

```
cnt_t nil_ready_all (
              void * p,
              cnt_t cnt,
              msg_t msg)
```

Puts in ready state all thread matching the specified status and associated object.

Parameters

in	*p*	object pointer
in	*cnt*	number of threads to be readied as a negative number, non negative numbers are ignored
in	*msg*	the wakeup message

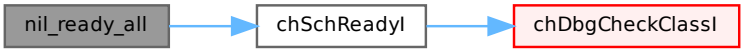
Returns

The number of readied threads.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.2.3.4.3 __dbg_check_disable()

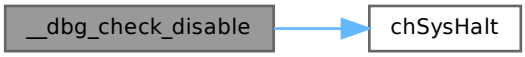
```
void __dbg_check_disable (
              void )
```

Guard code for chSysDisable().

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.2.3.4.4 __dbg_check_suspend()

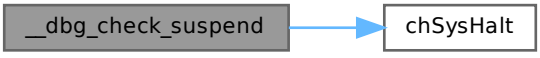
```
void __dbg_check_suspend (
              void )
```

Guard code for chSysSuspend().

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.2.3.4.5 __dbg_check_enable()

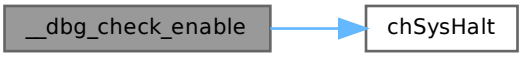
```
void __dbg_check_enable (
              void )
```

Guard code for chSysEnable().

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.2.3.4.6 __dbg_check_lock()

```
void __dbg_check_lock (
              void )
```

Guard code for chSysLock().

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

6.2.3.4.7 __dbg_check_unlock()

```
void __dbg_check_unlock (
              void )
```

Guard code for `chSysUnlock()`.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

6.2.3.4.8 __dbg_check_lock_from_isr()

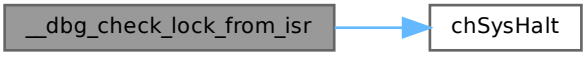
```
void __dbg_check_lock_from_isr (
              void )
```

Guard code for `chSysLockFromIsr()`.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.2.3.4.9 __dbg_check_unlock_from_isr()

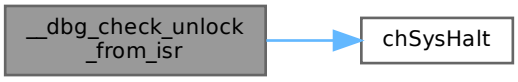
```
void __dbg_check_unlock_from_isr (
              void )
```

Guard code for `chSysUnlockFromIsr()`.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.2.3.4.10 __dbg_check_enter_isr()

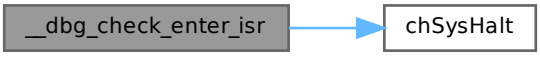
```
void __dbg_check_enter_isr (
              void )
```

Guard code for `CH_IRQ_PROLOGUE()`.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.2.3.4.11 __dbg_check_leave_isr()

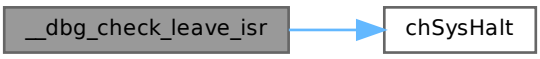
```
void __dbg_check_leave_isr (
              void )
```

Guard code for CH_IRQ_EPILOGUE().

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.2.3.4.12 chDbgCheckClassI()

```
void chDbgCheckClassI (
              void )
```

I-class functions context check.

Verifies that the system is in an appropriate state for invoking an I-class API function. A panic is generated if the state is not compatible.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.2.3.4.13 chDbgCheckClassS()

```
void chDbgCheckClassS (
             void )
```

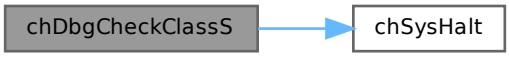
S-class functions context check.

Verifies that the system is in an appropriate state for invoking an S-class API function. A panic is generated if the state is not compatible.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.2.3.4.14 chSysInit()

```
void chSysInit (
             void )
```

Initializes the kernel.

Initializes the kernel structures, the current instructions flow becomes the idle thread upon return. The idle thread must not invoke any kernel primitive able to change state to not runnable.

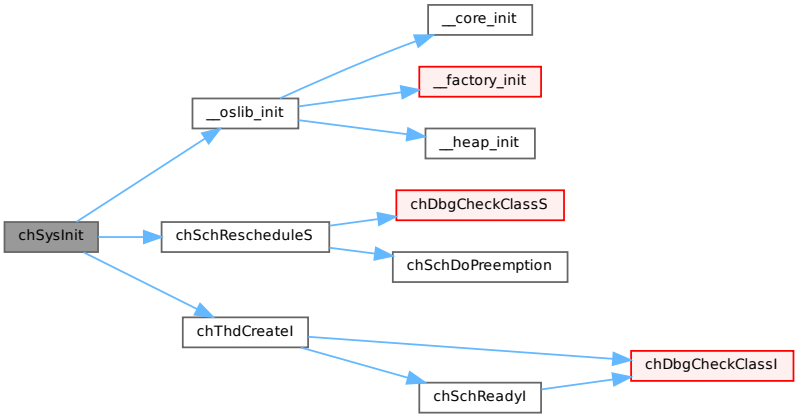
Note

This function assumes that the `nil` global variable has been zeroed by the runtime environment. If this is not the case then make sure to clear it before calling this function.

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



6.2.3.4.15 chSysHalt()

```
void chSysHalt (
            const char * reason)
```

Halts the system.

This function is invoked by the operating system when an unrecoverable error is detected, for example because a programming error in the application code that triggers an assertion while in debug mode.

Note

Can be invoked from any system state.

Parameters

in	*reason*	pointer to an error string

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.4.16 chSysTimerHandlerI()

```
void chSysTimerHandlerI (
            void )
```

Time management handler.

Note

This handler has to be invoked by a periodic ISR in order to reschedule the waiting threads.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.2.3.4.17 chSysUnconditionalLock()

```
void chSysUnconditionalLock (
            void )
```

Unconditionally enters the kernel lock state.

Note

Can be called without previous knowledge of the current lock state. The final state is "s-locked".

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.4.18 chSysUnconditionalUnlock()

```
void chSysUnconditionalUnlock (
            void )
```

Unconditionally leaves the kernel lock state.

Note

Can be called without previous knowledge of the current lock state. The final state is "normal".

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.4.19 chSysGetStatusAndLockX()

```
syssts_t chSysGetStatusAndLockX (
            void )
```

Returns the execution status and enters a critical zone.

This functions enters into a critical zone and can be called from any context. Because its flexibility it is less efficient than `chSysLock()` which is preferable when the calling context is known.

Postcondition

The system is in a critical zone.

Note

This function is only available if the underlying port supports `port_get_lock_status()` and `port_is_locked()`.

Returns

The previous system status, the encoding of this status word is architecture-dependent and opaque.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.2.3.4.20 chSysRestoreStatusX()

```
void chSysRestoreStatusX (
            syssts_t sts)
```

Restores the specified execution status and leaves a critical zone.

Note

A call to `chSchRescheduleS()` is automatically performed if exiting the critical zone and if not in ISR context.

This function is only available if the underlying port supports `port_get_lock_status()` and `port_is_locked()`.

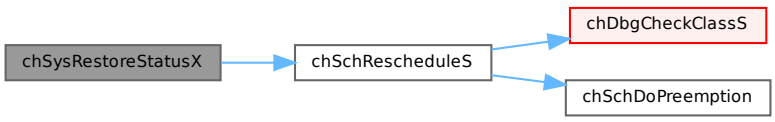
Parameters

in	*sts*	the system status to be restored.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

Here is the call graph for this function:



6.2.3.4.21 chSysIsCounterWithinX()

```
bool chSysIsCounterWithinX (
              rtcnt_t cnt,
              rtcnt_t start,
              rtcnt_t end)
```

Realtime window test.

This function verifies if the current realtime counter value lies within the specified range or not. The test takes care of the realtime counter wrapping to zero on overflow.

Note

When start==end then the function returns always false because a null time range is specified.

This function is only available if the port layer supports the option PORT_SUPPORTS_RT.

Parameters

in	*cnt*	the counter value to be tested
in	*start*	the start of the time window (inclusive)
in	*end*	the end of the time window (non inclusive)

Return values

true	current time within the specified time window.
false	current time not within the specified time window.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.2.3.4.22 chSysPolledDelayX()

```
void chSysPolledDelayX (
              rtcnt_t cycles)
```

Polled delay.

Note

The real delay is always few cycles in excess of the specified value.

This function is only available if the port layer supports the option PORT_SUPPORTS_RT.

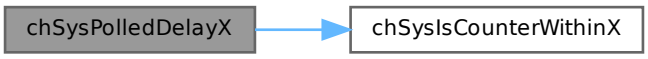
Parameters

in	*cycles*	number of cycles

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

Here is the call graph for this function:



6.2.3.4.23 chSchReadyI()

```
thread_t * chSchReadyI (
             thread_t * tp,
             msg_t msg)
```

Makes the specified thread ready for execution.

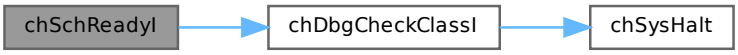
Parameters

in	*tp*	pointer to the thread_t object
in	*msg*	the wakeup message

Returns

The same reference passed as parameter.

Here is the call graph for this function:



6.2.3.4.24 chSchIsPreemptionRequired()

```
bool chSchIsPreemptionRequired (
             void )
```

Evaluates if preemption is required.

The decision is taken by comparing the relative priorities and depending on the state of the round robin timeout counter.

Note

Not a user function, it is meant to be invoked by the scheduler itself or from within the port layer.

Return values

true	if there is a thread that must go in running state immediately.
false	if preemption is not required.

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.4.25 chSchDoPreemption()

```
void chSchDoPreemption (
              void )
```

Switches to the first thread on the runnable queue.

Note

Not a user function, it is meant to be invoked by the scheduler itself or from within the port layer.

Function Class:

Special function, this function has special requirements see the notes.

6.2.3.4.26 chSchRescheduleS()

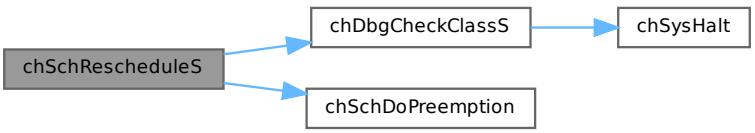
```
void chSchRescheduleS (
              void )
```

Reschedules if needed.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.2.3.4.27 chSchGoSleepTimeoutS()

```
msg_t chSchGoSleepTimeoutS (
              tstate_t newstate,
              sysinterval_t timeout)
```

Puts the current thread to sleep into the specified state with timeout specification.

The thread goes into a sleeping state, if it is not awakened explicitly within the specified system time then it is forcibly awakened with a `MSG_TIMEOUT` low level message.

Parameters

in	*newstate*	the new thread state or a semaphore pointer
in	*timeout*	the number of ticks before the operation timeouts. the following special values are allowed: • *TIME_INFINITE* no timeout.

Returns

The wakeup message.

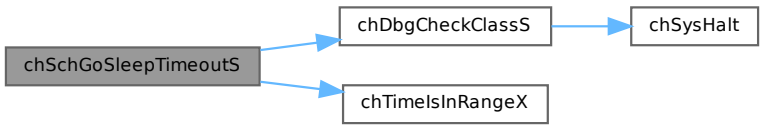
Return values

MSG_TIMEOUT	if a timeout occurred.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.2.3.4.28 chTimeIsInRangeX()

```
bool chTimeIsInRangeX (
             systime_t time,
             systime_t start,
             systime_t end)
```

Checks if the specified time is within the specified time range.

Note

When start==end then the function returns always false because the time window has zero size.

Parameters

in	*time*	the time to be verified
in	*start*	the start of the time window (inclusive)
in	*end*	the end of the time window (non inclusive)

Return values

true	current time within the specified time window.
false	current time not within the specified time window.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.2.3.4.29 chThdCreateI()

```
thread_t * chThdCreateI (
             const thread_descriptor_t * tdp)
```

Creates a new thread into a static memory area.

The new thread is initialized and make ready to execute.

Note

A thread can terminate by calling `chThdExit()` or by simply returning from its main function.

Parameters

`out`	*tdp*	pointer to the thread descriptor structure

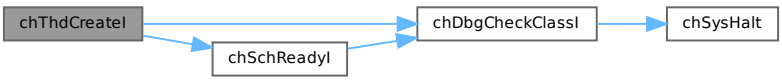
Returns

The pointer to the `thread_t` structure allocated for the thread.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.2.3.4.30 chThdCreate()

```
thread_t * chThdCreate (
             const thread_descriptor_t * tdp)
```

Creates a new thread into a static memory area.

The new thread is initialized and make ready to execute.

Note

A thread can terminate by calling `chThdExit()` or by simply returning from its main function.

Parameters

out	*tdp*	pointer to the thread descriptor structure

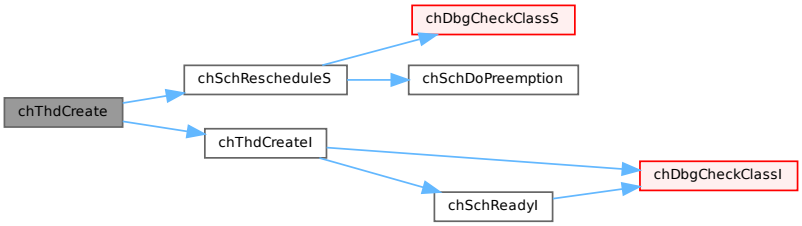
Returns

The pointer to the `thread_t` structure allocated for the thread.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.2.3.4.31 chThdExit()

```
void chThdExit (
            msg_t msg)
```

Terminates the current thread.

The thread goes in the `CH_STATE_FINAL` state holding the specified exit status code, other threads can retrieve the exit status code by invoking the function `chThdWait()`.

Postcondition

Eventual code after this function will never be executed, this function never returns. The compiler has no way to know this so do not assume that the compiler would remove the dead code.

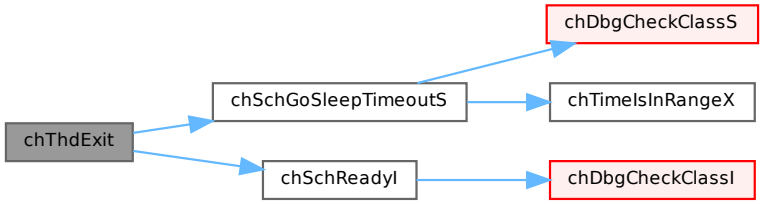
Parameters

in	*msg*	thread exit code

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.2.3.4.32 chThdWait()

```
msg_t chThdWait (
              thread_t * tp)
```

Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.

Parameters

in	*tp*	pointer to the thread

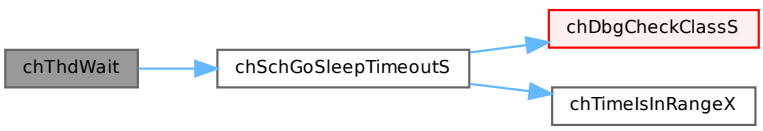
Returns

The exit code from the terminated thread.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.2.3.4.33 chThdSuspendTimeoutS()

```
msg_t chThdSuspendTimeoutS (
            thread_reference_t * trp,
            sysinterval_t timeout)
```

Sends the current thread sleeping and sets a reference variable.

Note

This function must reschedule, it can only be called from thread context.

Parameters

in	*trp*	a pointer to a thread reference object
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

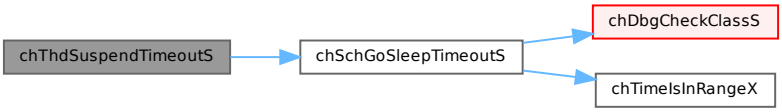
Returns

The wake up message.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.2.3.4.34 chThdResumeI()

```
void chThdResumeI (
            thread_reference_t * trp,
            msg_t msg)
```

Wakes up a thread waiting on a thread reference object.

Note

This function must not reschedule because it can be called from ISR context.

Parameters

in	*trp*	a pointer to a thread reference object
in	*msg*	the message code

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.2.3.4.35 chThdResume()

```
void chThdResume (
             thread_reference_t * trp,
             msg_t msg)
```

Wakes up a thread waiting on a thread reference object.

Note

This function must reschedule, it can only be called from thread context.

Parameters

in	*trp*	a pointer to a thread reference object
in	*msg*	the message code

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.4.36 chThdSleep()

```
void chThdSleep (
             sysinterval_t timeout)
```

Suspends the invoking thread for the specified time.

Parameters

in	*timeout*	the delay in system ticks

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.4.37 chThdSleepUntil()

```
void chThdSleepUntil (
              systime_t abstime)
```

Suspends the invoking thread until the system time arrives to the specified value.

Parameters

in	*abstime*	absolute system time

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.3.4.38 chThdEnqueueTimeoutS()

```
msg_t chThdEnqueueTimeoutS (
              threads_queue_t * tqp,
              sysinterval_t timeout)
```

Enqueues the caller thread on a threads queue object.

The caller thread is enqueued and put to sleep until it is dequeued or the specified timeouts expires.

Parameters

in	*tqp*	pointer to a `threads_queue_t` structure
in	*timeout*	the timeout in system ticks, the special values are handled as follow: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The message from `osalQueueWakeupOneI()` or `osalQueueWakeupAllI()` functions.

Return values

MSG_TIMEOUT	if the thread has not been dequeued within the specified timeout or if the function has been invoked with `TIME_IMMEDIATE` as timeout specification.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

6.2.3.4.39 chThdDoDequeueNextI()

```
void chThdDoDequeueNextI (
              threads_queue_t * tqp,
              msg_t msg)
```

Dequeues and wakes up one thread from the threads queue object.

Dequeues one thread from the queue without checking if the queue is empty.

Precondition

The queue must contain at least an object.

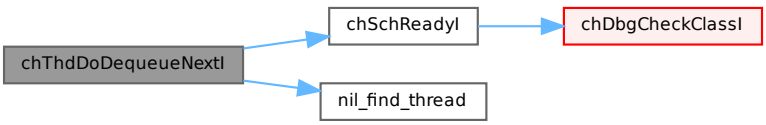
Parameters

`in`	*tqp*	pointer to a `threads_queue_t` structure
`in`	*msg*	the message code

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.2.3.4.40 chThdDequeueNextI()

```
void chThdDequeueNextI (
            threads_queue_t * tqp,
            msg_t msg)
```

Dequeues and wakes up one thread from the threads queue object, if any.

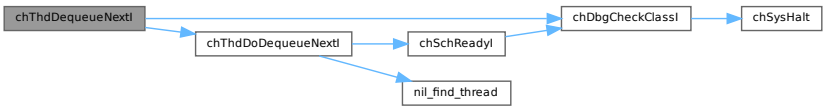
Parameters

in	*tqp*	pointer to a threads_queue_t structure
in	*msg*	the message code

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.2.3.4.41 chThdDequeueAllI()

```
void chThdDequeueAllI (
            threads_queue_t * tqp,
            msg_t msg)
```

Dequeues and wakes up all threads from the threads queue object.

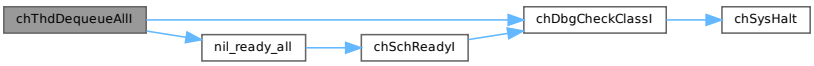
Parameters

in	*tqp*	pointer to a threads_queue_t structure
in	*msg*	the message code

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.2.3.5 Variable Documentation

6.2.3.5.1 nil

```
os_instance_t nil
```

System data structures.

6.2.4 Semaphores

6.2.4.1 Detailed Description

Semaphores macros

- #define __SEMAPHORE_DATA(name, n)
  Data part of a static semaphore initializer.
- #define SEMAPHORE_DECL(name, n)
  Static semaphore initializer.

Macro Functions

- #define chSemObjectInit(sp, n)
  Initializes a semaphore with the specified counter value.
- #define chSemReset(sp, n)
  Performs a reset operation on the semaphore.
- #define chSemResetI(sp, n)
  Performs a reset operation on the semaphore.
- #define chSemWait(sp)
  Performs a wait operation on a semaphore.
- #define chSemWaitS(sp)
  Performs a wait operation on a semaphore.
- #define chSemFastWaitI(sp)
  Decreases the semaphore counter.
- #define chSemFastSignalI(sp)
  Increases the semaphore counter.
- #define chSemGetCounterI(sp)
  Returns the semaphore counter current value.

Functions

- msg_t chSemWaitTimeout (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- msg_t chSemWaitTimeoutS (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- void chSemSignal (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemSignalI (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemResetWithMessage (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
- void chSemResetWithMessageI (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.

6.2.4.2 Macro Definition Documentation

6.2.4.2.1 __SEMAPHORE_DATA

```
#define __SEMAPHORE_DATA(
              name,
              n)
```

Value:

```
{n}
```

Data part of a static semaphore initializer.

This macro should be used when statically initializing a semaphore that is part of a bigger structure.

Parameters

in	*name*	the name of the semaphore variable
in	*n*	the counter initial value, this value must be non-negative

6.2.4.2.2 SEMAPHORE_DECL

```
#define SEMAPHORE_DECL(
              name,
              n)
```

Value:

```
semaphore_t name = __SEMAPHORE_DATA(name, n)
```

Static semaphore initializer.

Statically initialized semaphores require no explicit initialization using `chSemInit()`.

Parameters

in	*name*	the name of the semaphore variable
in	*n*	the counter initial value, this value must be non-negative

6.2.4.2.3 chSemObjectInit

```
#define chSemObjectInit(
              sp,
              n)
```

Value:

```
((sp)->cnt = (n))
```

Initializes a semaphore with the specified counter value.

Parameters

out	*sp*	pointer to a semaphore_t structure
in	*n*	initial value of the semaphore counter. Must be non-negative.

Function Class:

Object or module nitializer function.

6.2.4.2.4 chSemReset

```
#define chSemReset(
              sp,
              n)
```

Value:

```
chSemResetWithMessage(sp, n, MSG_RESET)
```

Performs a reset operation on the semaphore.

Postcondition

After invoking this function all the threads waiting on the semaphore, if any, are released and the semaphore counter is set to the specified, non negative, value.

Note

This function implicitly sends MSG_RESET as message.

Parameters

in	*sp*	pointer to a semaphore_t structure
in	*n*	the new value of the semaphore counter. The value must be non-negative.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.4.2.5 chSemResetI

```
#define chSemResetI(
              sp,
              n)
```

Value:

```
chSemResetWithMessageI(sp, n, MSG_RESET)
```

Performs a reset operation on the semaphore.

Postcondition

After invoking this function all the threads waiting on the semaphore, if any, are released and the semaphore counter is set to the specified, non negative, value.

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

Note

This function implicitly sends MSG_RESET as message.

Parameters

in	*sp*	pointer to a semaphore_t structure
in	*n*	the new value of the semaphore counter. The value must be non-negative.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.2.4.2.6 chSemWait

```
#define chSemWait(
                sp)
```

Value:

```
chSemWaitTimeout(sp, TIME_INFINITE)
```

Performs a wait operation on a semaphore.

Parameters

in	*sp*	pointer to a semaphore_t structure

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

CH_MSG_OK	if the thread has not stopped on the semaphore or the semaphore has been signaled.
CH_MSG_RST	if the semaphore has been reset using chSemReset().

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.4.2.7 chSemWaitS

```
#define chSemWaitS(
                sp)
```

Value:

```
chSemWaitTimeoutS(sp, TIME_INFINITE)
```

Performs a wait operation on a semaphore.

Parameters

in	*sp*	pointer to a semaphore_t structure

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

CH_MSG_OK	if the thread has not stopped on the semaphore or the semaphore has been signaled.
CH_MSG_RST	if the semaphore has been reset using chSemReset().

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

6.2.4.2.8 chSemFastWaitI

```
#define chSemFastWaitI(
              sp)
```

Value:

```
((sp)->cnt--)
```

Decreases the semaphore counter.

This macro can be used when the counter is known to be positive.

Parameters

in	*sp*	pointer to a semaphore_t structure

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.2.4.2.9 chSemFastSignalI

```
#define chSemFastSignalI(
              sp)
```

Value:

```
((sp)->cnt++)
```

Increases the semaphore counter.

This macro can be used when the counter is known to be not negative.

Parameters

in	*sp*	pointer to a semaphore_t structure

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.2.4.2.10 chSemGetCounterI

```
#define chSemGetCounterI(
              sp)
```

Value:

```
((sp)->cnt)
```

Returns the semaphore counter current value.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.2.4.3 Function Documentation

6.2.4.3.1 chSemWaitTimeout()

```
msg_t chSemWaitTimeout (
             semaphore_t * sp,
             sysinterval_t timeout)
```

Performs a wait operation on a semaphore with timeout specification.

Parameters

in	*sp*	pointer to a semaphore_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

NIL_MSG_OK	if the thread has not stopped on the semaphore or the semaphore has been signaled.
NIL_MSG_RST	if the semaphore has been reset using `chSemReset()`.
NIL_MSG_TMO	if the semaphore has not been signaled or reset within the specified timeout.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.2.4.3.2 chSemWaitTimeoutS()

```
msg_t chSemWaitTimeoutS (
              semaphore_t * sp,
              sysinterval_t timeout)
```

Performs a wait operation on a semaphore with timeout specification.

Parameters

in	*sp*	pointer to a `semaphore_t` structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

NIL_MSG_OK	if the thread has not stopped on the semaphore or the semaphore has been signaled.
NIL_MSG_RST	if the semaphore has been reset using `chSemReset()`.
NIL_MSG_TMO	if the semaphore has not been signaled or reset within the specified timeout.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

6.2.4.3.3 chSemSignal()

```
void chSemSignal (
            semaphore_t * sp)
```

Performs a signal operation on a semaphore.

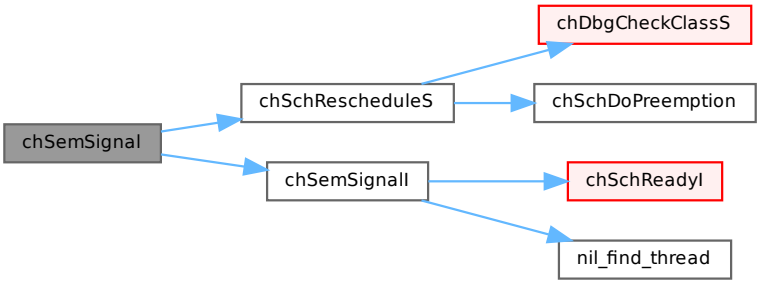
Parameters

in	*sp*	pointer to a semaphore_t structure

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.2.4.3.4 chSemSignalI()

```
void chSemSignalI (
             semaphore_t * sp)
```

Performs a signal operation on a semaphore.

Postcondition

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

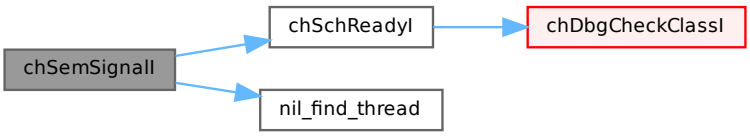
Parameters

in	*sp*	pointer to a semaphore_t structure

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.2.4.3.5 chSemResetWithMessage()

```
void chSemResetWithMessage (
             semaphore_t * sp,
             cnt_t n,
             msg_t msg)
```

Performs a reset operation on the semaphore.

Postcondition

After invoking this function all the threads waiting on the semaphore, if any, are released and the semaphore counter is set to the specified, non negative, value.

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

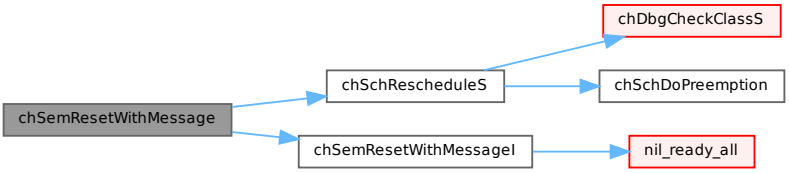
Parameters

in	*sp*	pointer to a semaphore_t structure
in	*n*	the new value of the semaphore counter. The value must be non-negative.
in	*msg*	message to be sent

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.2.4.3.6 chSemResetWithMessageI()

```
void chSemResetWithMessageI (
              semaphore_t * sp,
              cnt_t n,
              msg_t msg)
```

Performs a reset operation on the semaphore.

Postcondition

After invoking this function all the threads waiting on the semaphore, if any, are released and the semaphore counter is set to the specified, non negative, value.

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

Parameters

in	*sp*	pointer to a semaphore_t structure
in	*n*	the new value of the semaphore counter. The value must be non-negative.
in	*msg*	message to be sent

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.2.5 Events

6.2.5.1 Detailed Description

Macro Functions

- #define chEvtObjectInit(esp)

  Initializes an Event Source.
- #define chEvtRegisterMask(esp, elp, events)

  Registers an Event Listener on an Event Source.
- #define chEvtRegister(esp, elp, event)

  Registers an Event Listener on an Event Source.
- #define chEvtIsListeningI(esp)

  Verifies if there is at least one `event_listener_t` registered.
- #define chEvtBroadcast(esp)

  Signals all the Event Listeners registered on the specified Event Source.
- #define chEvtBroadcastI(esp)

  Signals all the Event Listeners registered on the specified Event Source.
- #define chEvtAddEventsI(events)

  *Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.*
- #define chEvtGetEventsX(void)

  Returns the events mask.
- #define chEvtWaitOne(events)

  Waits for exactly one of the specified events.
- #define chEvtWaitAny(events)

  Waits for any of the specified events.
- #define chEvtWaitAll(events)

  Waits for all the specified events.

Data Structures

- struct event_listener

  Event Listener structure.
- struct event_source

  Event Source structure.

Macros

- #define ALL_EVENTS ((eventmask_t)-1)

  All events allowed mask.
- #define EVENT_MASK(eid)

  Returns an event mask from an event identifier.
- #define _EVENTSOURCE_DATA(name)

  Data part of a static event source initializer.
- #define EVENTSOURCE_DECL(name)

  Static event source initializer.

Typedefs

- typedef struct event_listener event_listener_t
- typedef struct event_source event_source_t

  Event Source structure.
- typedef void(∗ evhandler_t) (eventid_t id)

  Event Handler callback function.

Functions

- void chEvtRegisterMaskWithFlags (event_source_t ∗esp, event_listener_t ∗elp, eventmask_t events, eventflags_t wflags)

  Registers an Event Listener on an Event Source.
- void chEvtUnregister (event_source_t ∗esp, event_listener_t ∗elp)

  Unregisters an Event Listener from its Event Source.
- eventmask_t chEvtGetAndClearEventsI (eventmask_t events)

  Clears the pending events specified in the events mask.
- eventmask_t chEvtGetAndClearEvents (eventmask_t events)

  Clears the pending events specified in the events mask.
- eventmask_t chEvtAddEvents (eventmask_t events)

  *Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.*
- void chEvtBroadcastFlagsI (event_source_t ∗esp, eventflags_t flags)

  Signals all the Event Listeners registered on the specified Event Source.
- eventflags_t chEvtGetAndClearFlags (event_listener_t ∗elp)

  Returns the flags associated to an `event_listener_t`.
- void chEvtSignal (thread_t ∗tp, eventmask_t events)

  Adds a set of event flags directly to the specified `thread_t`.
- void chEvtSignalI (thread_t ∗tp, eventmask_t events)

  Adds a set of event flags directly to the specified `thread_t`.
- void chEvtBroadcastFlags (event_source_t ∗esp, eventflags_t flags)

  Signals all the Event Listeners registered on the specified Event Source.
- eventflags_t chEvtGetAndClearFlagsI (event_listener_t ∗elp)

  Returns the unmasked flags associated to an `event_listener_t`.
- void chEvtDispatch (const evhandler_t ∗handlers, eventmask_t events)

  Invokes the event handlers associated to an event flags mask.
- eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)

  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, sysinterval_t timeout)

  Waits for any of the specified events.
- eventmask_t chEvtWaitAllTimeout (eventmask_t mask, sysinterval_t timeout)

  Waits for all the specified events.

6.2.5.2 Macro Definition Documentation

6.2.5.2.1 ALL_EVENTS

```
#define ALL_EVENTS ((eventmask_t)-1)
```

All events allowed mask.

6.2.5.2.2 EVENT_MASK

```
#define EVENT_MASK(
                eid)
```

Value:

```
((eventmask_t)1 « (eventmask_t)(eid))
```

Returns an event mask from an event identifier.

6.2.5.2.3 _EVENTSOURCE_DATA

```
#define _EVENTSOURCE_DATA(
                name)
```

Value:

```
{(event_listener_t *)(&name)}
```

Data part of a static event source initializer.

This macro should be used when statically initializing an event source that is part of a bigger structure.

Parameters

name	the name of the event source variable

6.2.5.2.4 EVENTSOURCE_DECL

```
#define EVENTSOURCE_DECL(
                name)
```

Value:

```
event_source_t name = _EVENTSOURCE_DATA(name)
```

Static event source initializer.

Statically initialized event sources require no explicit initialization using `chEvtInit()`.

Parameters

name	the name of the event source variable

6.2.5.2.5 chEvtObjectInit

```
#define chEvtObjectInit(
              esp)
```

Value:

```
  do {                                                \
  (esp)->next = (event_listener_t *)(esp);                                \
} while (0)
```

Initializes an Event Source.

Note

This function can be invoked before the kernel is initialized because it just prepares a event_source_t structure.

Parameters

in	*esp*	pointer to the event_source_t structure

Function Class:

Object or module nitializer function.

6.2.5.2.6 chEvtRegisterMask

```
#define chEvtRegisterMask(
              esp,
              elp,
              events)
```

Value:

```
  chEvtRegisterMaskWithFlags(esp, elp, events, (eventflags_t)-1)
```

Registers an Event Listener on an Event Source.

Once a thread has registered as listener on an event source it will be notified of all events broadcasted there.

Note

Multiple Event Listeners can specify the same bits to be ORed to different threads.

Parameters

in	*esp*	pointer to the event_source_t structure
out	*elp*	pointer to the event_listener_t structure
in	*events*	the mask of events to be ORed to the thread when the event source is broadcasted

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.5.2.7 chEvtRegister

```
#define chEvtRegister(
              esp,
              elp,
              event)
```

Value:

```
chEvtRegisterMask(esp, elp, EVENT_MASK(event))
```

Registers an Event Listener on an Event Source.

Note

Multiple Event Listeners can use the same event identifier, the listener will share the callback function.

Parameters

in	*esp*	pointer to the event_source_t structure
out	*elp*	pointer to the event_listener_t structure
in	*event*	numeric identifier assigned to the Event Listener. The value must range between zero and the size, in bit, of the eventmask_t type minus one.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.5.2.8 chEvtIsListeningI

```
#define chEvtIsListeningI(
              esp)
```

Value:

```
(bool)((esp) != (event_source_t *)(esp)->next)
```

Verifies if there is at least one event_listener_t registered.

Parameters

in	*esp*	pointer to the event_source_t structure

Returns

The event source status.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.2.5.2.9 chEvtBroadcast

```
#define chEvtBroadcast(
              esp)
```

Value:

```
chEvtBroadcastFlags(esp, (eventflags_t)0)
```

Signals all the Event Listeners registered on the specified Event Source.

Parameters

in	*esp*	pointer to the event_source_t structure

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.5.2.10 chEvtBroadcastI

```
#define chEvtBroadcastI(
              esp)
```

Value:

```
chEvtBroadcastFlagsI(esp, (eventflags_t)0)
```

Signals all the Event Listeners registered on the specified Event Source.

Postcondition

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

Parameters

in	*esp*	pointer to the event_source_t structure

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.2.5.2.11 chEvtAddEventsI

```
#define chEvtAddEventsI(
              events)
```

Value:

```
(nil.current->epmask |= events)
```

Adds (OR) a set of events to the current thread, this is **much** faster than using chEvtBroadcast() or chEvtSignal().

Parameters

in	*events*	the events to be added

Returns

The mask of currently pending events.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.2.5.2.12 chEvtGetEventsX

```
#define chEvtGetEventsX(
              void)
```

Value:

```
(nil.current->epmask)
```

Returns the events mask.

The pending events mask is returned but not altered in any way.

Returns

The pending events mask.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.5.2.13 chEvtWaitOne

```
#define chEvtWaitOne(
              events)
```

Value:

```
chEvtWaitOneTimeout(events, TIME_INFINITE)
```

Waits for exactly one of the specified events.

The function waits for one event among those specified in `events` to become pending then the event is cleared and returned.

Note

One and only one event is served in the function, the one with the lowest event id. The function is meant to be invoked into a loop in order to serve all the pending events.
This means that Event Listeners with a lower event identifier have an higher priority.

Parameters

in	*events*	events that the function should wait for, `ALL_EVENTS` enables all the events

Returns

The mask of the lowest event id served and cleared.

Return values

0	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.5.2.14 chEvtWaitAny

```
#define chEvtWaitAny(
              events)
```

Value:

```
chEvtWaitAnyTimeout(events, TIME_INFINITE)
```

Waits for any of the specified events.

The function waits for any event among those specified in `mask` to become pending then the events are cleared and returned.

Parameters

`in`	*events*	events that the function should wait for, `ALL_EVENTS` enables all the events

Returns

The mask of the served and cleared events.

Return values

0	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.5.2.15 chEvtWaitAll

```
#define chEvtWaitAll(
              events)
```

Value:

```
chEvtWaitAllTimeout(events, TIME_INFINITE)
```

Waits for all the specified events.

The function waits for all the events specified in `mask` to become pending then the events are cleared and returned.

Parameters

`in`	*events*	events that the function should wait for, `ALL_EVENTS` enables all the events

Returns

The mask of the served and cleared events.

Return values

0	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.5.3 Typedef Documentation

6.2.5.3.1 event_listener_t

```
typedef struct event_listener event_listener_t
```

6.2.5.3.2 event_source_t

```
typedef struct event_source event_source_t
```

Event Source structure.

6.2.5.3.3 evhandler_t

```
typedef void(* evhandler_t) (eventid_t id)
```

Event Handler callback function.

6.2.5.4 Function Documentation

6.2.5.4.1 chEvtRegisterMaskWithFlags()

```
void chEvtRegisterMaskWithFlags (
             event_source_t * esp,
             event_listener_t * elp,
             eventmask_t events,
             eventflags_t wflags)
```

Registers an Event Listener on an Event Source.

Once a thread has registered as listener on an event source it will be notified of all events broadcasted there.

Note

Multiple Event Listeners can specify the same bits to be ORed to different threads.

Parameters

in	*esp*	pointer to the event_source_t structure
in	*elp*	pointer to the event_listener_t structure
in	*events*	events to be ORed to the thread when the event source is broadcasted
in	*wflags*	mask of flags the listening thread is interested in

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.5.4.2 chEvtUnregister()

```
void chEvtUnregister (
             event_source_t * esp,
             event_listener_t * elp)
```

Unregisters an Event Listener from its Event Source.

Note

If the event listener is not registered on the specified event source then the function does nothing.

For optimal performance it is better to perform the unregister operations in inverse order of the register operations (elements are found on top of the list).

Parameters

in	*esp*	pointer to the event_source_t structure
in	*elp*	pointer to the event_listener_t structure

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.5.4.3 chEvtGetAndClearEventsI()

```
eventmask_t chEvtGetAndClearEventsI (
             eventmask_t events)
```

Clears the pending events specified in the events mask.

Parameters

in	*events*	the events to be cleared

Returns

The mask of pending events that were cleared.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.2.5.4.4 chEvtGetAndClearEvents()

```
eventmask_t chEvtGetAndClearEvents (
              eventmask_t events)
```

Clears the pending events specified in the events mask.

Parameters

in	*events*	the events to be cleared

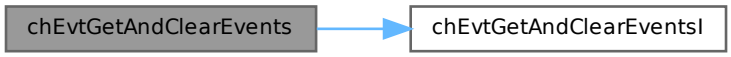
Returns

The mask of pending events that were cleared.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.2.5.4.5 chEvtAddEvents()

```
eventmask_t chEvtAddEvents (
              eventmask_t events)
```

Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.

Parameters

in	*events*	the events to be added

Returns

The mask of currently pending events.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.5.4.6 chEvtBroadcastFlagsI()

```
void chEvtBroadcastFlagsI (
            event_source_t * esp,
            eventflags_t flags)
```

Signals all the Event Listeners registered on the specified Event Source.

This function variants ORs the specified event flags to all the threads registered on the event_source_t in addition to the event flags specified by the threads themselves in the event_listener_t objects.

Postcondition

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

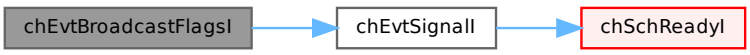
Parameters

in	*esp*	pointer to the event_source_t structure
in	*flags*	the flags set to be added to the listener flags mask

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.2.5.4.7 chEvtGetAndClearFlags()

```
eventflags_t chEvtGetAndClearFlags (
            event_listener_t * elp)
```

Returns the flags associated to an event_listener_t.

The flags are returned and the event_listener_t flags mask is cleared.

Parameters

in	*elp*	pointer to the event_listener_t structure

Returns

The flags added to the listener by the associated event source.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.5.4.8 chEvtSignal()

```
void chEvtSignal (
             thread_t * tp,
             eventmask_t events)
```

Adds a set of event flags directly to the specified thread_t.

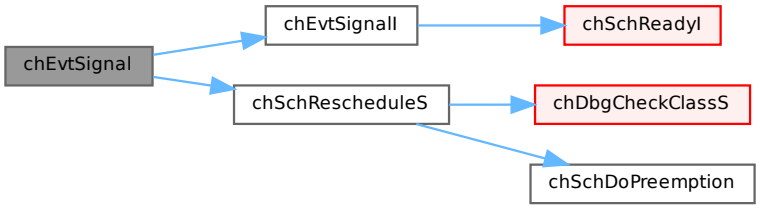
Parameters

in	*tp*	the thread to be signaled
in	*events*	the event flags set to be ORed

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.2.5.4.9 chEvtSignalI()

```
void chEvtSignalI (
             thread_t * tp,
             eventmask_t events)
```

Adds a set of event flags directly to the specified thread_t.

Postcondition

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

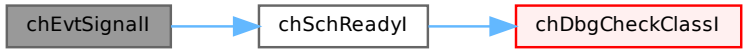
Parameters

in	*tp*	the thread to be signaled
in	*events*	the event flags set to be ORed

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.2.5.4.10 chEvtBroadcastFlags()

```
void chEvtBroadcastFlags (
            event_source_t * esp,
            eventflags_t flags)
```

Signals all the Event Listeners registered on the specified Event Source.

This function variants ORs the specified event flags to all the threads registered on the event_source_t in addition to the event flags specified by the threads themselves in the event_listener_t objects.

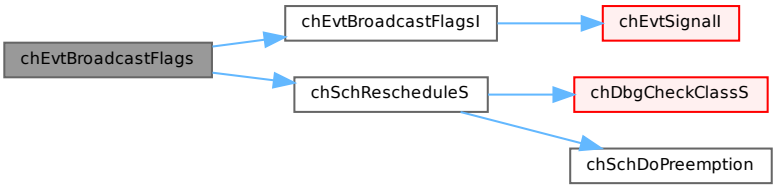
Parameters

in	*esp*	pointer to the event_source_t structure
in	*flags*	the flags set to be added to the listener flags mask

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.2.5.4.11 chEvtGetAndClearFlagsI()

```
eventflags_t chEvtGetAndClearFlagsI (
            event_listener_t * elp)
```

Returns the unmasked flags associated to an event_listener_t.

The flags are returned and the event_listener_t flags mask is cleared.

Parameters

in	*elp*	pointer to the event_listener_t structure

Returns

The flags added to the listener by the associated event source.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.2.5.4.12 chEvtDispatch()

```
void chEvtDispatch (
              const evhandler_t * handlers,
              eventmask_t events)
```

Invokes the event handlers associated to an event flags mask.

Parameters

in	*events*	mask of events to be dispatched
in	*handlers*	an array of evhandler_t. The array must have size equal to the number of bits in eventmask_t.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.5.4.13 chEvtWaitOneTimeout()

```
eventmask_t chEvtWaitOneTimeout (
              eventmask_t events,
              sysinterval_t timeout)
```

Waits for exactly one of the specified events.

The function waits for one event among those specified in `events` to become pending then the event is cleared and returned.

Note

One and only one event is served in the function, the one with the lowest event id. The function is meant to be invoked into a loop in order to serve all the pending events.
This means that Event Listeners with a lower event identifier have an higher priority.

Parameters

in	*events*	events that the function should wait for, ALL_EVENTS enables all the events
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The mask of the lowest event id served and cleared.

Return values

0	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.2.5.4.14 chEvtWaitAnyTimeout()

```
eventmask_t chEvtWaitAnyTimeout (
            eventmask_t mask,
            sysinterval_t timeout)
```

Waits for any of the specified events.

The function waits for any event among those specified in mask to become pending then the events are cleared and returned.

Parameters

in	*mask*	mask of the event flags that the function should wait for, ALL_EVENTS enables all the events
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The mask of the served and cleared events.

Return values

0	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.2.5.4.15 chEvtWaitAllTimeout()

```
eventmask_t chEvtWaitAllTimeout (
              eventmask_t mask,
              sysinterval_t timeout)
```

Waits for all the specified events.

The function waits for all the events specified in `mask` to become pending then the events are cleared and returned.

Parameters

`in`	*mask*	mask of the event flags that the function should wait for, `ALL_EVENTS` enables all the events
`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The mask of the served and cleared events.

Return values

0	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.2.6 Synchronous Messages

6.2.6.1 Detailed Description

Macro Functions

- #define chMsgWaitS()

  Suspends the thread and waits for an incoming message.
- #define chMsgGet(tp)

  Returns the message carried by the specified thread.
- #define chMsgReleaseS(tp, msg)

  Releases the thread waiting on top of the messages queue.

Functions

- msg_t chMsgSend (thread_t *tp, msg_t msg)

  Sends a message to the specified thread.
- thread_t * chMsgWait (void)

  Suspends the thread and waits for an incoming message.
- thread_t * chMsgWaitTimeout (sysinterval_t timeout)

  Suspends the thread and waits for an incoming message or a timeout to occur.
- thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)

  Suspends the thread and waits for an incoming message or a timeout to occur.
- void chMsgRelease (thread_t *tp, msg_t msg)

  Releases a sender thread specifying a response message.

6.2.6.2 Macro Definition Documentation

6.2.6.2.1 chMsgWaitS

```
#define chMsgWaitS()
```

Value:

```
chMsgWaitTimeoutS(TIME_INFINITE)
```

Suspends the thread and waits for an incoming message.

Postcondition

After receiving a message the function `chMsgGet()` must be called in order to retrieve the message and then `chMsgRelease()` must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke `chMsgRelease()` because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Returns

A pointer to the thread carrying the message.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

6.2.6.2.2 chMsgGet

```
#define chMsgGet(
              tp)
```

Value:

```
((tp)->sntmsg)
```

Returns the message carried by the specified thread.

Precondition

This function must be invoked immediately after exiting a call to `chMsgWait()`.

Parameters

in	*tp*	pointer to the thread

Returns

The message carried by the sender.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.6.2.3 chMsgReleaseS

```
#define chMsgReleaseS(
              tp,
              msg)
```

Value:

```
  do {                                          \
  (void) chSchReadyI(tp, msg);                                  \
  chSchRescheduleS();                                           \
  } while (false)
```

Releases the thread waiting on top of the messages queue.

Precondition

Invoke this function only after a message has been received using `chMsgWait()`.

Parameters

in	*tp*	pointer to the thread
in	*msg*	message to be returned to the sender

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

6.2.6.3 Function Documentation

6.2.6.3.1 chMsgSend()

```
msg_t chMsgSend (
             thread_t * tp,
             msg_t msg)
```

Sends a message to the specified thread.

The sender is stopped until the receiver executes a `chMsgRelease()` `after` receiving the message.

Parameters

in	*tp*	the pointer to the thread
in	*msg*	the message

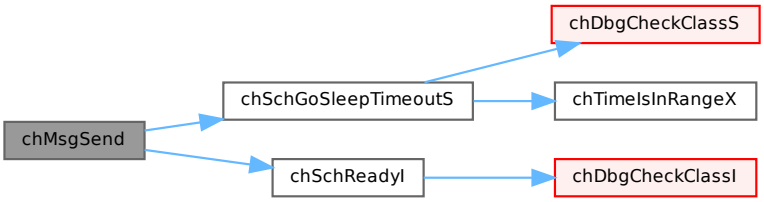
Returns

The answer message from `chMsgRelease()`.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.2.6.3.2 chMsgWait()

```
thread_t * chMsgWait (
             void )
```

Suspends the thread and waits for an incoming message.

Postcondition

After receiving a message the function `chMsgGet()` must be called in order to retrieve the message and then `chMsgRelease()` must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke `chMsgRelease()` because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Returns

A pointer to the thread carrying the message.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.2.6.3.3 chMsgWaitTimeout()

```
thread_t * chMsgWaitTimeout (
             sysinterval_t timeout)
```

Suspends the thread and waits for an incoming message or a timeout to occur.

Postcondition

After receiving a message the function `chMsgGet()` must be called in order to retrieve the message and then `chMsgRelease()` must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke `chMsgRelease()` because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Parameters

in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

A pointer to the thread carrying the message.

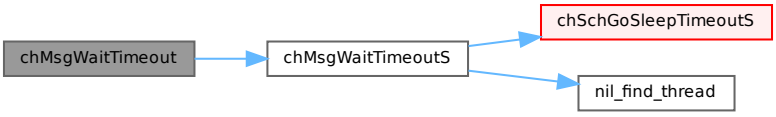
Return values

NULL	if a timeout occurred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.2.6.3.4 chMsgWaitTimeoutS()

```
thread_t * chMsgWaitTimeoutS (
             sysinterval_t timeout)
```

Suspends the thread and waits for an incoming message or a timeout to occur.

Postcondition

After receiving a message the function `chMsgGet()` must be called in order to retrieve the message and then `chMsgRelease()` must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke `chMsgRelease()` because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Parameters

in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_INFINITE* no timeout.

Returns

A pointer to the thread carrying the message.

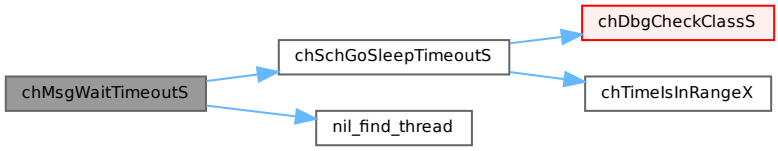
Return values

NULL	if a timeout occurred.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.2.6.3.5 chMsgRelease()

```
void chMsgRelease (
            thread_t * tp,
            msg_t msg)
```

Releases a sender thread specifying a response message.

Precondition

Invoke this function only after a message has been received using `chMsgWait()`.

Parameters

in	*tp*	pointer to the thread
in	*msg*	message to be returned to the sender

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3 OS Library

6.3.1 Detailed Description

The OS Library is a set of RTOS extensions compatible with both the RT and NIL RTOSes.

Topics

- Version Numbers and Identification
- Synchronization
- Memory Management
- Complex Services

6.3.2 Version Numbers and Identification

6.3.2.1 Detailed Description

OS Library related info.

ChibiOS/LIB version identification

- #define CH_OSLIB_VERSION "1.3.2"

  OS Library version string.
- #define CH_OSLIB_MAJOR 1

  OS Library version major number.
- #define CH_OSLIB_MINOR 3

  OS Library version minor number.
- #define CH_OSLIB_PATCH 2

  OS Library version patch number.

Macros

- #define __CHIBIOS_OSLIB__
  ChibiOS/LIB identification macro.
- #define CH_OSLIB_STABLE 1
  Stable release flag.
- #define CH_CFG_USE_FACTORY FALSE
- #define CH_CFG_USE_HEAP FALSE
- #define CH_CFG_USE_MEMPOOLS FALSE
- #define CH_CFG_USE_OBJ_FIFOS FALSE
- #define CH_CFG_USE_PIPES FALSE
- #define CH_CFG_USE_OBJ_CACHES FALSE
- #define CH_CFG_USE_DELEGATES FALSE
- #define CH_CFG_USE_JOBS FALSE
- #define CH_CFG_USE_MAILBOXES FALSE

Functions

- static void __oslib_init (void)
  Initialization of all library modules.

6.3.2.2 Macro Definition Documentation

6.3.2.2.1 __CHIBIOS_OSLIB__

```
#define __CHIBIOS_OSLIB__
```

ChibiOS/LIB identification macro.

6.3.2.2.2 CH_OSLIB_STABLE

```
#define CH_OSLIB_STABLE 1
```

Stable release flag.

6.3.2.2.3 CH_OSLIB_VERSION

```
#define CH_OSLIB_VERSION "1.3.2"
```

OS Library version string.

6.3.2.2.4 CH_OSLIB_MAJOR

```
#define CH_OSLIB_MAJOR 1
```

OS Library version major number.

6.3.2.2.5 CH_OSLIB_MINOR

```
#define CH_OSLIB_MINOR 3
```

OS Library version minor number.

6.3.2.2.6 CH_OSLIB_PATCH

```
#define CH_OSLIB_PATCH 2
```

OS Library version patch number.

6.3.2.2.7 CH_CFG_USE_FACTORY

```
#define CH_CFG_USE_FACTORY FALSE
```

6.3.2.2.8 CH_CFG_USE_HEAP

```
#define CH_CFG_USE_HEAP FALSE
```

6.3.2.2.9 CH_CFG_USE_MEMPOOLS

```
#define CH_CFG_USE_MEMPOOLS FALSE
```

6.3.2.2.10 CH_CFG_USE_OBJ_FIFOS

```
#define CH_CFG_USE_OBJ_FIFOS FALSE
```

6.3.2.2.11 CH_CFG_USE_PIPES

```
#define CH_CFG_USE_PIPES FALSE
```

6.3.2.2.12 CH_CFG_USE_OBJ_CACHES

```
#define CH_CFG_USE_OBJ_CACHES FALSE
```

6.3.2.2.13 CH_CFG_USE_DELEGATES

```
#define CH_CFG_USE_DELEGATES FALSE
```

6.3.2.2.14 CH_CFG_USE_JOBS

```
#define CH_CFG_USE_JOBS FALSE
```

6.3.2.2.15 CH_CFG_USE_MAILBOXES

```
#define CH_CFG_USE_MAILBOXES FALSE
```

6.3.2.3 Function Documentation

6.3.2.3.1 __oslib_init()

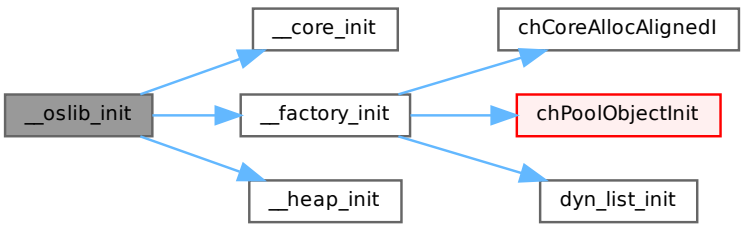
```
void __oslib_init (
              void )  [inline], [static]
```

Initialization of all library modules.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.3.3 Synchronization

6.3.3.1 Detailed Description

Synchronization services.

Topics

- Binary Semaphores
- Mailboxes
- Pipes
- Delegate Threads
- Jobs Queues

6.3.3.2 Binary Semaphores

6.3.3.2.1 Detailed Description

Data Structures

- struct ch_binary_semaphore

  Binary semaphore type.

Macros

- #define __BSEMAPHORE_DATA(name, taken)

  Data part of a static semaphore initializer.
- #define BSEMAPHORE_DECL(name, taken)

  Static semaphore initializer.

Typedefs

- typedef struct ch_binary_semaphore binary_semaphore_t

  Binary semaphore type.

Functions

- static void chBSemObjectInit (binary_semaphore_t *bsp, bool taken)

  Initializes a binary semaphore.
- static msg_t chBSemWait (binary_semaphore_t *bsp)

  Wait operation on the binary semaphore.
- static msg_t chBSemWaitS (binary_semaphore_t *bsp)

  Wait operation on the binary semaphore.
- static msg_t chBSemWaitTimeoutS (binary_semaphore_t *bsp, sysinterval_t timeout)

  Wait operation on the binary semaphore.
- static msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, sysinterval_t timeout)

  Wait operation on the binary semaphore.
- static void chBSemResetI (binary_semaphore_t *bsp, bool taken)

  Reset operation on the binary semaphore.
- static void chBSemReset (binary_semaphore_t *bsp, bool taken)

  Reset operation on the binary semaphore.
- static void chBSemSignalI (binary_semaphore_t *bsp)

  Performs a signal operation on a binary semaphore.
- static void chBSemSignal (binary_semaphore_t *bsp)

  Performs a signal operation on a binary semaphore.
- static bool chBSemGetStateI (const binary_semaphore_t *bsp)

  Returns the binary semaphore current state.

6.3.3.2.2 Macro Definition Documentation

6.3.3.2.2.1 __BSEMAPHORE_DATA

```
#define __BSEMAPHORE_DATA(
              name,
              taken)
```

Value:

```
{__SEMAPHORE_DATA(name.sem, ((taken) ? 0 : 1))}
```

Data part of a static semaphore initializer.

This macro should be used when statically initializing a semaphore that is part of a bigger structure.

Parameters

in	*name*	the name of the semaphore variable
in	*taken*	the semaphore initial state

6.3.3.2.2.2 BSEMAPHORE_DECL

```
#define BSEMAPHORE_DECL(
              name,
              taken)
```

Value:

```
binary_semaphore_t name = __BSEMAPHORE_DATA(name, taken)
```

Static semaphore initializer.

Statically initialized semaphores require no explicit initialization using `chBSemInit()`.

Parameters

in	*name*	the name of the semaphore variable
in	*taken*	the semaphore initial state

6.3.3.2.3 Typedef Documentation

6.3.3.2.3.1 binary_semaphore_t

```
typedef struct ch_binary_semaphore binary_semaphore_t
```

Binary semaphore type.

6.3.3.2.4 Function Documentation

6.3.3.2.4.1 chBSemObjectInit()

```
void chBSemObjectInit (
             binary_semaphore_t * bsp,
             bool taken)  [inline], [static]
```

Initializes a binary semaphore.

Parameters

out	*bsp*	pointer to a `binary_semaphore_t` structure
in	*taken*	initial state of the binary semaphore: • *false*, the initial state is not taken. • *true*, the initial state is taken.

Function Class:

Object or module nitializer function.

6.3.3.2.4.2 chBSemWait()

```
msg_t chBSemWait (
            binary_semaphore_t * bsp)  [inline], [static]
```

Wait operation on the binary semaphore.

Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

MSG_OK	if the binary semaphore has been successfully taken.
MSG_RESET	if the binary semaphore has been reset using `chBSemReset()`.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.3.2.4.3 chBSemWaitS()

```
msg_t chBSemWaitS (
            binary_semaphore_t * bsp)  [inline], [static]
```

Wait operation on the binary semaphore.

Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

MSG_OK	if the binary semaphore has been successfully taken.
MSG_RESET	if the binary semaphore has been reset using `chBSemReset()`.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

6.3.3.2.4.4 chBSemWaitTimeoutS()

```
msg_t chBSemWaitTimeoutS (
            binary_semaphore_t * bsp,
            sysinterval_t timeout)  [inline], [static]
```

Wait operation on the binary semaphore.

Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

MSG_OK	if the binary semaphore has been successfully taken.
MSG_RESET	if the binary semaphore has been reset using `chBSemReset()`.
MSG_TIMEOUT	if the binary semaphore has not been signaled or reset within the specified timeout.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

6.3.3.2.4.5 chBSemWaitTimeout()

```
msg_t chBSemWaitTimeout (
            binary_semaphore_t * bsp,
            sysinterval_t timeout)  [inline], [static]
```

Wait operation on the binary semaphore.

Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

MSG_OK	if the binary semaphore has been successfully taken.
MSG_RESET	if the binary semaphore has been reset using `chBSemReset()`.
MSG_TIMEOUT	if the binary semaphore has not been signaled or reset within the specified timeout.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.3.2.4.6 chBSemResetI()

```
void chBSemResetI (
            binary_semaphore_t * bsp,
            bool taken)  [inline], [static]
```

Reset operation on the binary semaphore.

Note

The released threads can recognize they were waked up by a reset rather than a signal because the `chBSemWait()` will return `MSG_RESET` instead of `MSG_OK`.

This function does not reschedule.

Parameters

`in`	*bsp*	pointer to a `binary_semaphore_t` structure
`in`	*taken*	new state of the binary semaphore • *false*, the new state is not taken. • *true*, the new state is taken.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.3.2.4.7 chBSemReset()

```
void chBSemReset (
            binary_semaphore_t * bsp,
            bool taken)  [inline], [static]
```

Reset operation on the binary semaphore.

Note

The released threads can recognize they were waked up by a reset rather than a signal because the `chBSemWait()` will return `MSG_RESET` instead of `MSG_OK`.

Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure
in	*taken*	new state of the binary semaphore • *false*, the new state is not taken. • *true*, the new state is taken.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.3.2.4.8 chBSemSignalI()

```
void chBSemSignalI (
            binary_semaphore_t * bsp)  [inline], [static]
```

Performs a signal operation on a binary semaphore.

Note

This function does not reschedule.

Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.3.3.2.4.9 chBSemSignal()

```
void chBSemSignal (
            binary_semaphore_t * bsp)  [inline], [static]
```

Performs a signal operation on a binary semaphore.

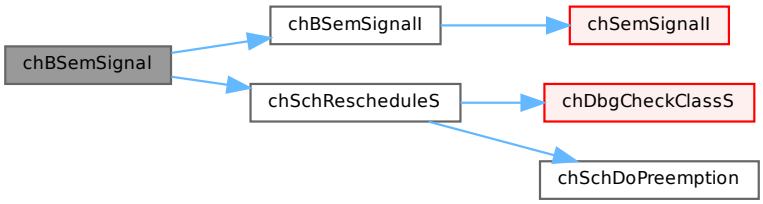
Parameters

in	*bsp*	pointer to a binary_semaphore_t structure

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.2.4.10 chBSemGetStateI()

```
bool chBSemGetStateI (
            const binary_semaphore_t * bsp)  [inline], [static]
```

Returns the binary semaphore current state.

Parameters

in	*bsp*	pointer to a binary_semaphore_t structure

Returns

The binary semaphore current state.

Return values

false	if the binary semaphore is not taken.
true	if the binary semaphore is taken.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.3.3 Mailboxes

6.3.3.3.1 Detailed Description

Asynchronous messages.

Operation mode

A mailbox is an asynchronous communication mechanism.
Operations defined for mailboxes:

- **Post**: Posts a message on the mailbox in FIFO order.
- **Post Ahead**: Posts a message on the mailbox with urgent priority.
- **Fetch**: A message is fetched from the mailbox and removed from the queue.
- **Reset**: The mailbox is emptied and all the stored messages are lost.

A message is a variable of type msg_t that is guaranteed to have the same size of and be compatible with (data) pointers (anyway an explicit cast is needed). If larger messages need to be exchanged then a pointer to a structure can be posted in the mailbox but the posting side has no predefined way to know when the message has been processed. A possible approach is to allocate memory (from a memory pool for example) from the posting side and free it on the fetching side. Another approach is to set a "done" flag into the structure pointed by the message.

Precondition

In order to use the mailboxes APIs the `CH_CFG_USE_MAILBOXES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

Data Structures

- struct mailbox_t

  Structure representing a mailbox object.

Macros

- #define __MAILBOX_DATA(name, buffer, size)

  Data part of a static mailbox initializer.
- #define MAILBOX_DECL(name, buffer, size)

  Static mailbox initializer.

Functions

- void chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)

  Initializes a `mailbox_t` object.
- void chMBReset (mailbox_t *mbp)

  Resets a `mailbox_t` object.
- void chMBResetI (mailbox_t *mbp)

  Resets a `mailbox_t` object.
- msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts a message into a mailbox.
- msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts a message into a mailbox.
- msg_t chMBPostI (mailbox_t *mbp, msg_t msg)

  Posts a message into a mailbox.
- msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)

  Posts an high priority message into a mailbox.
- msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)

  Retrieves a message from a mailbox.
- msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)

  Retrieves a message from a mailbox.
- msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)

  Retrieves a message from a mailbox.
- static size_t chMBGetSizeI (const mailbox_t *mbp)

  Returns the mailbox buffer size as number of messages.
- static size_t chMBGetUsedCountI (const mailbox_t *mbp)

  Returns the number of used message slots into a mailbox.
- static size_t chMBGetFreeCountI (const mailbox_t *mbp)

  Returns the number of free message slots into a mailbox.
- static msg_t chMBPeekI (const mailbox_t *mbp)

  Returns the next message in the queue without removing it.
- static void chMBResumeX (mailbox_t *mbp)

  Terminates the reset state.

6.3.3.3.2 Macro Definition Documentation

6.3.3.3.2.1 __MAILBOX_DATA

```
#define __MAILBOX_DATA(
              name,
              buffer,
              size)
```

Value:

```
  {                                 \
  (msg_t *)(buffer),                                    \
  (msg_t *)(buffer) + size,                             \
  (msg_t *)(buffer),                                    \
```

```
  (msg_t *)(buffer),                                                    \
  (size_t)0,                                                            \
  false,                                                                \
  __THREADS_QUEUE_DATA(name.qw),                                        \
  __THREADS_QUEUE_DATA(name.qr),                                        \
}
```

Data part of a static mailbox initializer.

This macro should be used when statically initializing a mailbox that is part of a bigger structure.

Parameters

in	*name*	the name of the mailbox variable
in	*buffer*	pointer to the mailbox buffer array of msg_t
in	*size*	number of msg_t elements in the buffer array

6.3.3.3.2.2 MAILBOX_DECL

```
#define MAILBOX_DECL(
              name,
              buffer,
              size)
```

Value:

```
  mailbox_t name = __MAILBOX_DATA(name, buffer, size)
```

Static mailbox initializer.

Statically initialized mailboxes require no explicit initialization using chMBObjectInit().

Parameters

in	*name*	the name of the mailbox variable
in	*buffer*	pointer to the mailbox buffer array of msg_t
in	*size*	number of msg_t elements in the buffer array

6.3.3.3.3 Function Documentation

6.3.3.3.3.1 chMBObjectInit()

```
void chMBObjectInit (
             mailbox_t * mbp,
             msg_t * buf,
             size_t n)
```

Initializes a mailbox_t object.

Parameters

out	*mbp*	the pointer to the mailbox_t structure to be initialized
in	*buf*	pointer to the messages buffer as an array of msg_t
in	*n*	number of elements in the buffer array

Function Class:

Object or module nitializer function.

6.3.3.3.3.2 chMBReset()

```
void chMBReset (
            mailbox_t * mbp)
```

Resets a mailbox_t object.

All the waiting threads are resumed with status MSG_RESET and the queued messages are lost.

Postcondition

The mailbox is in reset state, all operations will fail and return MSG_RESET until the mailbox is enabled again using chMBResumeX().

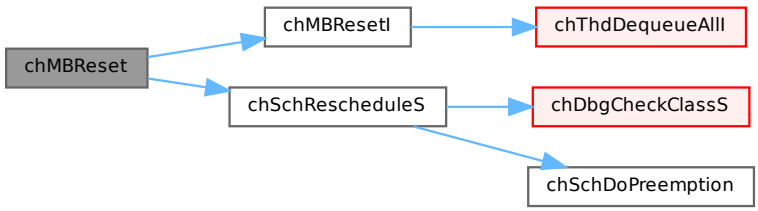
Parameters

in	*mbp*	the pointer to an initialized mailbox_t object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.3.3.3 chMBResetI()

```
void chMBResetI (
            mailbox_t * mbp)
```

Resets a mailbox_t object.

All the waiting threads are resumed with status MSG_RESET and the queued messages are lost.

Postcondition

The mailbox is in reset state, all operations will fail and return MSG_RESET until the mailbox is enabled again using chMBResumeX().

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.3.3.3.4 chMBPostTimeout()

```
msg_t chMBPostTimeout (
            mailbox_t * mbp,
            msg_t msg,
            sysinterval_t timeout)
```

Posts a message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object
in	*msg*	the message to be posted on the mailbox
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

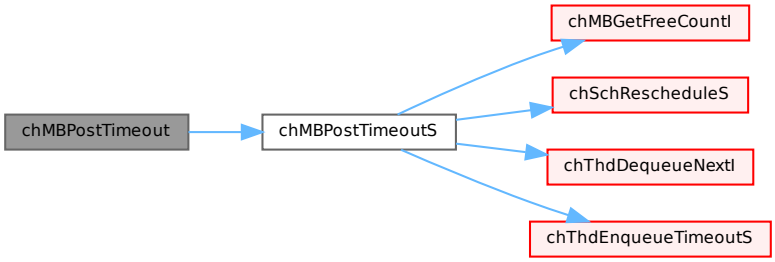
Return values

MSG_OK	if a message has been correctly posted.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.3.3.5 chMBPostTimeoutS()

```
msg_t chMBPostTimeoutS (
              mailbox_t * mbp,
              msg_t msg,
              sysinterval_t timeout)
```

Posts a message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object
in	*msg*	the message to be posted on the mailbox
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

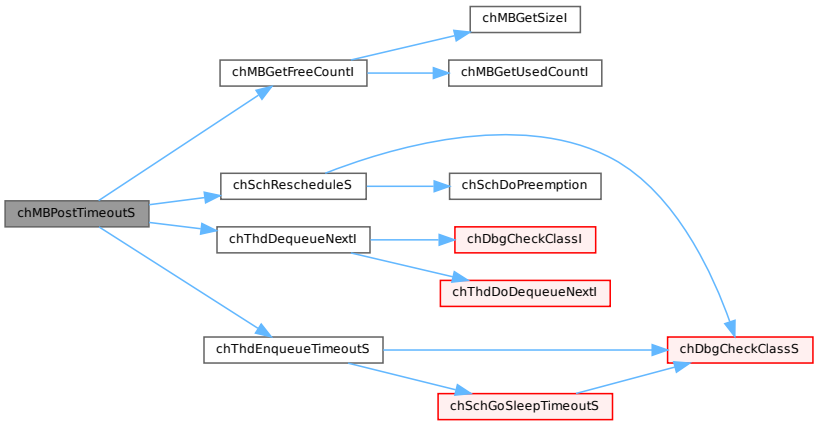
Return values

MSG_OK	if a message has been correctly posted.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.3.3.3.6 chMBPostI()

```
msg_t chMBPostI (
              mailbox_t * mbp,
              msg_t msg)
```

Posts a message into a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object
in	*msg*	the message to be posted on the mailbox

Returns

The operation status.

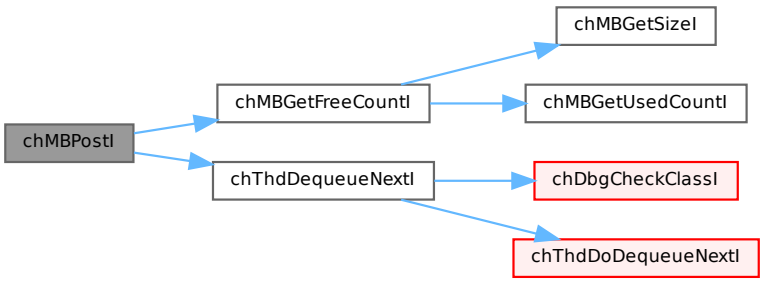
Return values

MSG_OK	if a message has been correctly posted.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the mailbox is full and the message cannot be posted.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.3.3.3.3.7 chMBPostAheadTimeout()

```
msg_t chMBPostAheadTimeout (
              mailbox_t * mbp,
              msg_t msg,
              sysinterval_t timeout)
```

Posts an high priority message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object
in	*msg*	the message to be posted on the mailbox
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

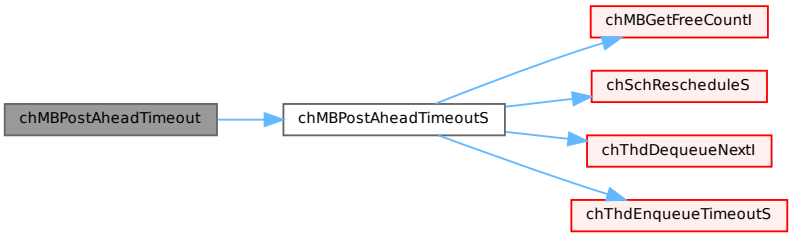
Return values

MSG_OK	if a message has been correctly posted.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.3.3.8 chMBPostAheadTimeoutS()

```
msg_t chMBPostAheadTimeoutS (
            mailbox_t * mbp,
            msg_t msg,
            sysinterval_t timeout)
```

Posts an high priority message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object
in	*msg*	the message to be posted on the mailbox
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

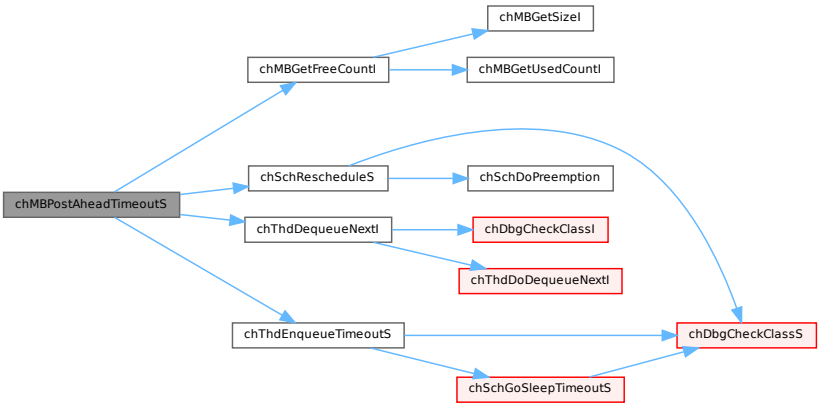
Return values

MSG_OK	if a message has been correctly posted.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.3.3.3.9 chMBPostAheadI()

```
msg_t chMBPostAheadI (
            mailbox_t * mbp,
            msg_t msg)
```

Posts an high priority message into a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object
in	*msg*	the message to be posted on the mailbox

Returns

The operation status.

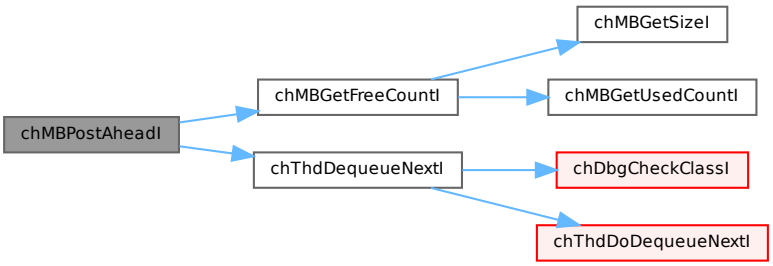
Return values

MSG_OK	if a message has been correctly posted.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the mailbox is full and the message cannot be posted.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.3.3.3.3.10 chMBFetchTimeout()

```
msg_t chMBFetchTimeout (
              mailbox_t * mbp,
              msg_t * msgp,
              sysinterval_t timeout)
```

Retrieves a message from a mailbox.

The invoking thread waits until a message is posted in the mailbox or the specified time runs out.

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object
out	*msgp*	pointer to a message variable for the received message
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

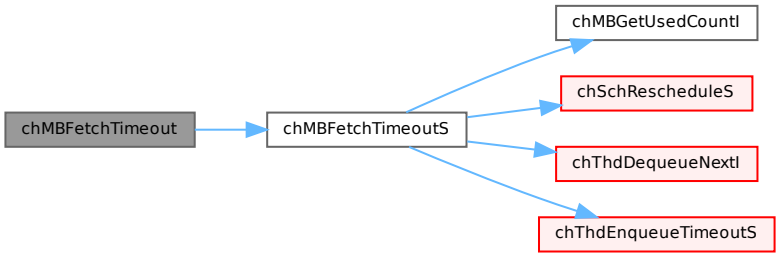
Return values

MSG_OK	if a message has been correctly fetched.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.3.3.11 chMBFetchTimeoutS()

```
msg_t chMBFetchTimeoutS (
            mailbox_t * mbp,
            msg_t * msgp,
            sysinterval_t timeout)
```

Retrieves a message from a mailbox.

The invoking thread waits until a message is posted in the mailbox or the specified time runs out.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object
out	*msgp*	pointer to a message variable for the received message
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

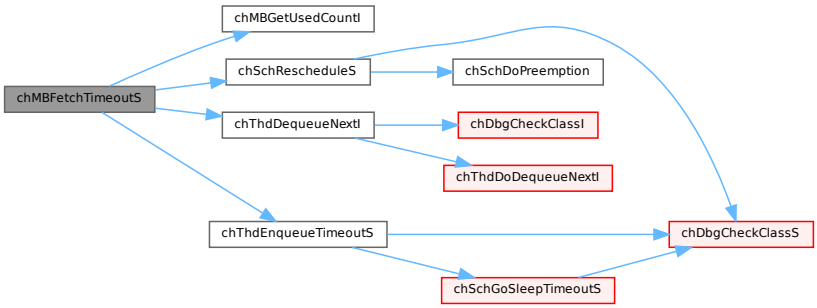
Return values

MSG_OK	if a message has been correctly fetched.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.3.3.3.12 chMBFetchI()

```
msg_t chMBFetchI (
              mailbox_t * mbp,
              msg_t * msgp)
```

Retrieves a message from a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is empty.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object
out	*msgp*	pointer to a message variable for the received message

Returns

The operation status.

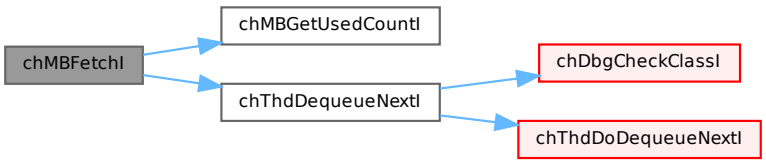
Return values

MSG_OK	if a message has been correctly fetched.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the mailbox is empty and a message cannot be fetched.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.3.3.3.3.13 chMBGetSizeI()

```
size_t chMBGetSizeI (
            const mailbox_t * mbp)  [inline], [static]
```

Returns the mailbox buffer size as number of messages.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object

Returns

The size of the mailbox.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.3.3.3.14 chMBGetUsedCountI()

```
size_t chMBGetUsedCountI (
            const mailbox_t * mbp)  [inline], [static]
```

Returns the number of used message slots into a mailbox.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object

Returns

The number of queued messages.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.3.3.3.15 chMBGetFreeCountI()

```
size_t chMBGetFreeCountI (
            const mailbox_t * mbp)  [inline], [static]
```

Returns the number of free message slots into a mailbox.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object

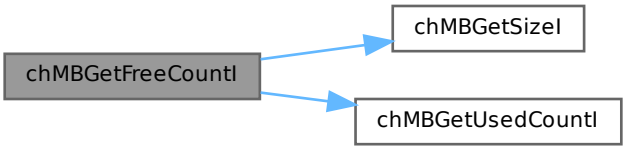
Returns

The number of empty message slots.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.3.3.3.3.16 chMBPeekI()

```
msg_t chMBPeekI (
            const mailbox_t * mbp)  [inline], [static]
```

Returns the next message in the queue without removing it.

Precondition

A message must be waiting in the queue for this function to work or it would return garbage. The correct way to use this macro is to use chMBGetUsedCountI() and then use this macro, all within a lock state.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object

Returns

The next message in queue.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.3.3.3.17 chMBResumeX()

```
void chMBResumeX (
             mailbox_t * mbp)  [inline], [static]
```

Terminates the reset state.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.3.3.4 Pipes

6.3.3.4.1 Detailed Description

Data Structures

- struct pipe_t

  Structure representing a pipe object.

Macros

- #define PC_INIT(p)
- #define PC_LOCK(p)
- #define PC_UNLOCK(p)
- #define PW_INIT(p)
- #define PW_LOCK(p)
- #define PW_UNLOCK(p)
- #define PR_INIT(p)
- #define PR_LOCK(p)
- #define PR_UNLOCK(p)
- #define __PIPE_DATA(name, buffer, size)

  Data part of a static pipe initializer.
- #define PIPE_DECL(name, buffer, size)

  Static pipe initializer.

Functions

- static size_t pipe_write (pipe_t *pp, const uint8_t *bp, size_t n)

  Non-blocking pipe write.
- static size_t pipe_read (pipe_t *pp, uint8_t *bp, size_t n)

  Non-blocking pipe read.
- void chPipeObjectInit (pipe_t *pp, uint8_t *buf, size_t n)

  Initializes a `mailbox_t` object.
- void chPipeReset (pipe_t *pp)

  Resets a `pipe_t` object.
- size_t chPipeWriteTimeout (pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe write with timeout.
- size_t chPipeReadTimeout (pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe read with timeout.
- static size_t chPipeGetSize (const pipe_t *pp)

  Returns the pipe buffer size as number of bytes.
- static size_t chPipeGetUsedCount (const pipe_t *pp)

  Returns the number of used byte slots into a pipe.
- static size_t chPipeGetFreeCount (const pipe_t *pp)

  Returns the number of free byte slots into a pipe.
- static void chPipeResume (pipe_t *pp)

  Terminates the reset state.

6.3.3.4.2 Macro Definition Documentation

6.3.3.4.2.1 PC_INIT

```
#define PC_INIT(
                p)
```

Value:

```
chMtxObjectInit(&(p)->cmtx)
```

6.3.3.4.2.2 PC_LOCK

```
#define PC_LOCK(
                p)
```

Value:

```
chMtxLock(&(p)->cmtx)
```

6.3.3.4.2.3 PC_UNLOCK

```
#define PC_UNLOCK(
                p)
```

Value:

```
chMtxUnlock(&(p)->cmtx)
```

6.3.3.4.2.4 PW_INIT

```
#define PW_INIT(
                p)
```

Value:

```
chMtxObjectInit(&(p)->wmtx)
```

6.3.3.4.2.5 PW_LOCK

```
#define PW_LOCK(
                p)
```

Value:

```
chMtxLock(&(p)->wmtx)
```

6.3.3.4.2.6 PW_UNLOCK

```
#define PW_UNLOCK(
                p)
```

Value:

```
chMtxUnlock(&(p)->wmtx)
```

6.3.3.4.2.7 PR_INIT

```
#define PR_INIT(
                p)
```

Value:

```
chMtxObjectInit(&(p)->rmtx)
```

6.3.3.4.2.8 PR_LOCK

```
#define PR_LOCK(
                p)
```

Value:

```
chMtxLock(&(p)->rmtx)
```

6.3.3.4.2.9 PR_UNLOCK

```
#define PR_UNLOCK(
                p)
```

Value:

```
chMtxUnlock(&(p)->rmtx)
```

6.3.3.4.2.10 __PIPE_DATA

```
#define __PIPE_DATA(
              name,
              buffer,
              size)
```

Value:

```
  {                                   \
  (uint8_t *)(buffer),                                    \
  (uint8_t *)(buffer) + size,                             \
  (uint8_t *)(buffer),                                    \
  (uint8_t *)(buffer),                                    \
  (size_t)0,                                              \
  false,                                                  \
  NULL,                                                   \
  NULL,                                                   \
  __MUTEX_DATA(name.cmtx),                                \
  __MUTEX_DATA(name.wmtx),                                \
  __MUTEX_DATA(name.rmtx),                                \
}
```

Data part of a static pipe initializer.

This macro should be used when statically initializing a pipe that is part of a bigger structure.

Parameters

in	*name*	the name of the pipe variable
in	*buffer*	pointer to the pipe buffer array of `uint8_t`
in	*size*	number of `uint8_t` elements in the buffer array

6.3.3.4.2.11 PIPE_DECL

```
#define PIPE_DECL(
              name,
              buffer,
              size)
```

Value:

```
  pipe_t name = __PIPE_DATA(name, buffer, size)
```

Static pipe initializer.

Statically initialized pipes require no explicit initialization using `chPipeObjectInit()`.

Parameters

in	*name*	the name of the pipe variable
in	*buffer*	pointer to the pipe buffer array of `uint8_t`
in	*size*	number of `uint8_t` elements in the buffer array

6.3.3.4.3 Function Documentation

6.3.3.4.3.1 pipe_write()

```
size_t pipe_write (
            pipe_t * pp,
            const uint8_t * bp,
            size_t n)  [static]
```

Non-blocking pipe write.

The function writes data from a buffer to a pipe. The operation completes when the specified amount of data has been transferred or when the pipe buffer has been filled.

Parameters

in	*pp*	the pointer to an initialized pipe_t object
in	*bp*	pointer to the data buffer
in	*n*	the maximum amount of data to be transferred, the value 0 is reserved

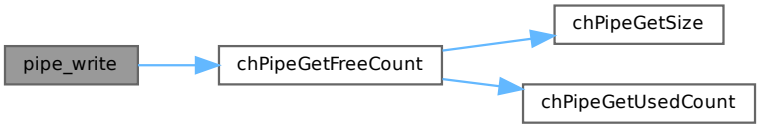
Returns

The number of bytes effectively transferred.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.3.3.4.3.2 pipe_read()

```
size_t pipe_read (
            pipe_t * pp,
            uint8_t * bp,
            size_t n)  [static]
```

Non-blocking pipe read.

The function reads data from a pipe into a buffer. The operation completes when the specified amount of data has been transferred or when the pipe buffer has been emptied.

Parameters

in	*pp*	the pointer to an initialized pipe_t object
out	*bp*	pointer to the data buffer
in	*n*	the maximum amount of data to be transferred, the value 0 is reserved

Returns

The number of bytes effectively transferred.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

6.3.3.4.3.3 chPipeObjectInit()

```
void chPipeObjectInit (
             pipe_t * pp,
             uint8_t * buf,
             size_t n)
```

Initializes a mailbox_t object.

Parameters

out	*pp*	the pointer to the pipe_t structure to be initialized
in	*buf*	pointer to the pipe buffer as an array of uint8_t
in	*n*	number of elements in the buffer array

Function Class:

Object or module nitializer function.

6.3.3.4.3.4 chPipeReset()

```
void chPipeReset (
             pipe_t * pp)
```

Resets a pipe_t object.

All the waiting threads are resumed with status MSG_RESET and the queued data is lost.

Postcondition

The pipe is in reset state, all operations will fail and return MSG_RESET until the mailbox is enabled again using chPipeResumeX().

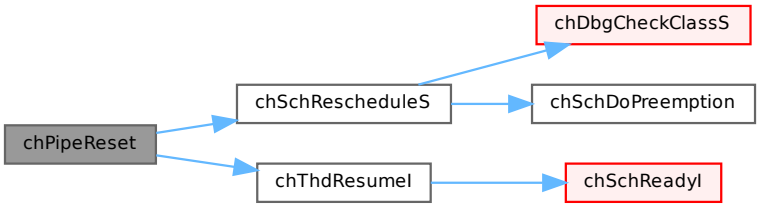
Parameters

in	*pp*	the pointer to an initialized pipe_t object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.4.3.5 chPipeWriteTimeout()

```
size_t chPipeWriteTimeout (
            pipe_t * pp,
            const uint8_t * bp,
            size_t n,
            sysinterval_t timeout)
```

Pipe write with timeout.

The function writes data from a buffer to a pipe. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the pipe has been reset.

Parameters

in	*pp*	the pointer to an initialized pipe_t object
in	*bp*	pointer to the data buffer
in	*n*	the number of bytes to be written, the value 0 is reserved
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

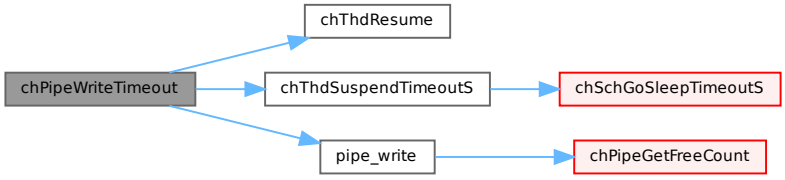
Returns

The number of bytes effectively transferred. A number lower than n means that a timeout occurred or the pipe went in reset state.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.4.3.6 chPipeReadTimeout()

```
size_t chPipeReadTimeout (
            pipe_t * pp,
            uint8_t * bp,
            size_t n,
            sysinterval_t timeout)
```

Pipe read with timeout.

The function reads data from a pipe into a buffer. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the pipe has been reset.

Parameters

in	*pp*	the pointer to an initialized pipe_t object
out	*bp*	pointer to the data buffer
in	*n*	the number of bytes to be read, the value 0 is reserved
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

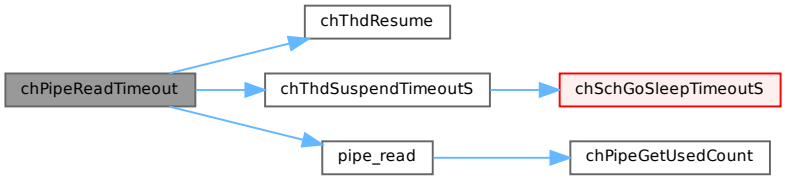
Returns

The number of bytes effectively transferred. A number lower than n means that a timeout occurred or the pipe went in reset state.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.4.3.7 chPipeGetSize()

```
size_t chPipeGetSize (
            const pipe_t * pp)  [inline], [static]
```

Returns the pipe buffer size as number of bytes.

Parameters

in	*pp*	the pointer to an initialized pipe_t object

Returns

The size of the pipe.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.3.4.3.8 chPipeGetUsedCount()

```
size_t chPipeGetUsedCount (
            const pipe_t * pp)  [inline], [static]
```

Returns the number of used byte slots into a pipe.

Parameters

in	*pp*	the pointer to an initialized pipe_t object

Returns

The number of queued bytes.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.3.4.3.9 chPipeGetFreeCount()

```
size_t chPipeGetFreeCount (
             const pipe_t * pp)  [inline], [static]
```

Returns the number of free byte slots into a pipe.

Parameters

in	*pp*	the pointer to an initialized pipe_t object

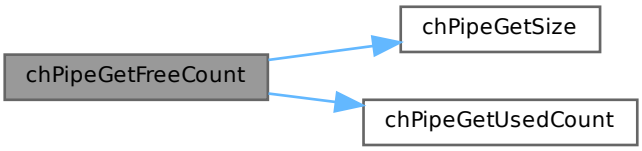
Returns

The number of empty byte slots.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.4.3.10 chPipeResume()

```
void chPipeResume (
             pipe_t * pp)  [inline], [static]
```

Terminates the reset state.

Parameters

in	*pp*	the pointer to an initialized pipe_t object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.3.5 Delegate Threads

6.3.3.5.1 Detailed Description

Typedefs

- typedef msg_t(∗ delegate_veneer_t) (va_list ∗argsp)

  Type of a delegate veneer function.
- typedef msg_t(∗ delegate_fn0_t) (void)

  Type of a delegate function with no parameters.
- typedef msg_t(∗ delegate_fn1_t) (msg_t p1)

  Type of a delegate function with one parameter.
- typedef msg_t(∗ delegate_fn2_t) (msg_t p1, msg_t p2)

  Type of a delegate function with two parameters.
- typedef msg_t(∗ delegate_fn3_t) (msg_t p1, msg_t p2, msg_t p3)

  Type of a delegate function with three parameters.
- typedef msg_t(∗ delegate_fn4_t) (msg_t p1, msg_t p2, msg_t p3, msg_t p4)

  Type of a delegate function with four parameters.

Functions

- msg_t __ch_delegate_fn0 (va_list ∗argsp)

  Veneer for functions with no parameters.
- msg_t __ch_delegate_fn1 (va_list ∗argsp)

  Veneer for functions with one parameter.
- msg_t __ch_delegate_fn2 (va_list ∗argsp)

  Veneer for functions with two parameters.
- msg_t __ch_delegate_fn3 (va_list ∗argsp)

  Veneer for functions with three parameters.
- msg_t __ch_delegate_fn4 (va_list ∗argsp)

  Veneer for functions with four parameters.
- msg_t chDelegateCallVeneer (thread_t ∗tp, delegate_veneer_t veneer,...)

  Triggers a function call on a delegate thread.
- void chDelegateDispatch (void)

  Call messages dispatching.
- msg_t chDelegateDispatchTimeout (sysinterval_t timeout)

  Call messages dispatching with timeout.
- static msg_t chDelegateCallDirect0 (thread_t ∗tp, delegate_fn0_t func)

  Direct call to a function with no parameters.
- static msg_t chDelegateCallDirect1 (thread_t ∗tp, delegate_fn1_t func, msg_t p1)

  Direct call to a function with one parameter.
- static msg_t chDelegateCallDirect2 (thread_t ∗tp, delegate_fn2_t func, msg_t p1, msg_t p2)

  Direct call to a function with two parameters.
- static msg_t chDelegateCallDirect3 (thread_t ∗tp, delegate_fn3_t func, msg_t p1, msg_t p2, msg_t p3)

  Direct call to a function with three parameters.
- static msg_t chDelegateCallDirect4 (thread_t ∗tp, delegate_fn4_t func, msg_t p1, msg_t p2, msg_t p3, msg_t p4)

  Direct call to a function with four parameters.

6.3.3.5.2 Typedef Documentation

6.3.3.5.2.1 delegate_veneer_t

```
typedef msg_t(* delegate_veneer_t) (va_list *argsp)
```

Type of a delegate veneer function.

6.3.3.5.2.2 delegate_fn0_t

```
typedef msg_t(* delegate_fn0_t) (void)
```

Type of a delegate function with no parameters.

6.3.3.5.2.3 delegate_fn1_t

```
typedef msg_t(* delegate_fn1_t) (msg_t p1)
```

Type of a delegate function with one parameter.

6.3.3.5.2.4 delegate_fn2_t

```
typedef msg_t(* delegate_fn2_t) (msg_t p1, msg_t p2)
```

Type of a delegate function with two parameters.

6.3.3.5.2.5 delegate_fn3_t

```
typedef msg_t(* delegate_fn3_t) (msg_t p1, msg_t p2, msg_t p3)
```

Type of a delegate function with three parameters.

6.3.3.5.2.6 delegate_fn4_t

```
typedef msg_t(* delegate_fn4_t) (msg_t p1, msg_t p2, msg_t p3, msg_t p4)
```

Type of a delegate function with four parameters.

6.3.3.5.3 Function Documentation

6.3.3.5.3.1 __ch_delegate_fn0()

```
msg_t __ch_delegate_fn0 (
            va_list * argsp)
```

Veneer for functions with no parameters.

Parameters

in	*argsp*	the list of arguments

Returns

The function return value.

6.3.3.5.3.2 __ch_delegate_fn1()

```
msg_t __ch_delegate_fn1 (
              va_list * argsp)
```

Veneer for functions with one parameter.

Parameters

in	*argsp*	the list of arguments

Returns

The function return value.

6.3.3.5.3.3 __ch_delegate_fn2()

```
msg_t __ch_delegate_fn2 (
              va_list * argsp)
```

Veneer for functions with two parameters.

Parameters

in	*argsp*	the list of arguments

Returns

The function return value.

6.3.3.5.3.4 __ch_delegate_fn3()

```
msg_t __ch_delegate_fn3 (
              va_list * argsp)
```

Veneer for functions with three parameters.

Parameters

in	*argsp*	the list of arguments

Returns

The function return value.

6.3.3.5.3.5 __ch_delegate_fn4()

```
msg_t __ch_delegate_fn4 (
            va_list * argsp)
```

Veneer for functions with four parameters.

Parameters

in	*argsp*	the list of arguments

Returns

The function return value.

6.3.3.5.3.6 chDelegateCallVeneer()

```
msg_t chDelegateCallVeneer (
            thread_t * tp,
            delegate_veneer_t veneer,
             ...)
```

Triggers a function call on a delegate thread.

Note

The thread must be executing `chDelegateDispatchTimeout()` in order to have the functions called.

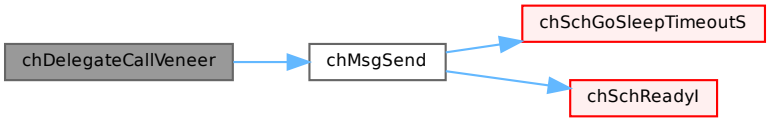
Parameters

in	*tp*	pointer to the delegate thread
in	*veneer*	pointer to the veneer function to be called
in	...	variable number of parameters

Returns

The function return value casted to msg_t. It is garbage for functions returning `void`.

Here is the call graph for this function:



6.3.3.5.3.7 chDelegateDispatch()

```
void chDelegateDispatch (
             void )
```

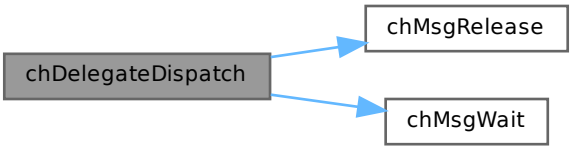
Call messages dispatching.

The function awaits for an incoming call messages and calls the specified functions, then it returns. In case multiple threads are sending messages then the requests are served in priority order.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.5.3.8 chDelegateDispatchTimeout()

```
msg_t chDelegateDispatchTimeout (
             sysinterval_t timeout)
```

Call messages dispatching with timeout.

The function awaits for an incoming call messages and calls the specified functions, then it returns. In case multiple threads are sending messages then the requests are served in priority order.

Parameters

in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_INFINITE* no timeout.

Returns

The function outcome.

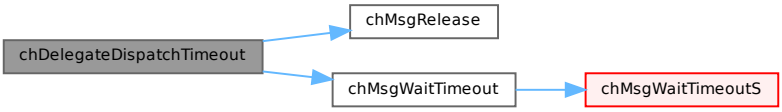
Return values

MSG_OK	if a function has been called.
MSG_TIMEOUT	if a timeout occurred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.5.3.9 chDelegateCallDirect0()

```
msg_t chDelegateCallDirect0 (
            thread_t * tp,
            delegate_fn0_t func)  [inline], [static]
```

Direct call to a function with no parameters.

Note

The return value is assumed to be not larger than a data pointer type. If you need a portable function then use `chDelegateCallVeneer()` instead.

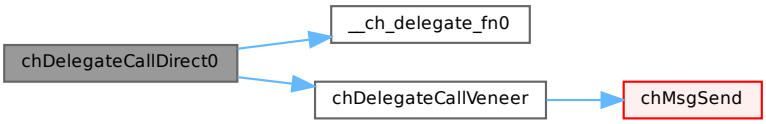
Parameters

in	*tp*	pointer to the delegate thread
in	*func*	pointer to the function to be called

Returns

The function return value as a `msg_t`.

Here is the call graph for this function:



6.3.3.5.3.10 chDelegateCallDirect1()

```
msg_t chDelegateCallDirect1 (
            thread_t * tp,
            delegate_fn1_t func,
            msg_t p1)  [inline], [static]
```

Direct call to a function with one parameter.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use `chDelegateCallVeneer()` instead.

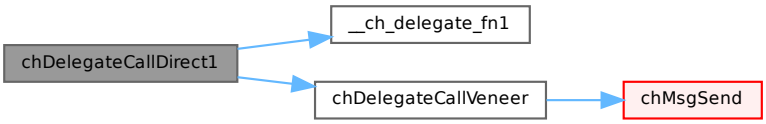
Parameters

in	*tp*	pointer to the delegate thread
in	*func*	pointer to the function to be called
in	*p1*	parameter 1 passed as a `msg_t`

Returns

The function return value as a `msg_t`.

Here is the call graph for this function:



6.3.3.5.3.11 chDelegateCallDirect2()

```
msg_t chDelegateCallDirect2 (
            thread_t * tp,
            delegate_fn2_t func,
            msg_t p1,
            msg_t p2)  [inline], [static]
```

Direct call to a function with two parameters.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use `chDelegateCallVeneer()` instead.

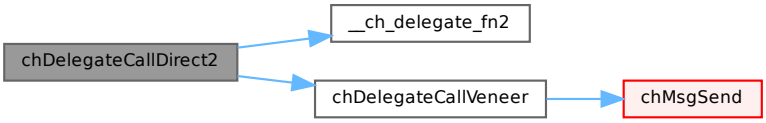
Parameters

in	*tp*	pointer to the delegate thread
in	*func*	pointer to the function to be called
in	*p1*	parameter 1 passed as a msg_t
in	*p2*	parameter 2 passed as a msg_t

Returns

The function return value as a msg_t.

Here is the call graph for this function:



6.3.3.5.3.12 chDelegateCallDirect3()

```
msg_t chDelegateCallDirect3 (
            thread_t * tp,
            delegate_fn3_t func,
            msg_t p1,
            msg_t p2,
            msg_t p3)  [inline], [static]
```

Direct call to a function with three parameters.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use chDelegateCallVeneer() instead.

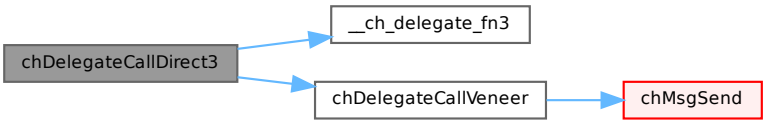
Parameters

in	*tp*	pointer to the delegate thread
in	*func*	pointer to the function to be called
in	*p1*	parameter 1 passed as a msg_t
in	*p2*	parameter 2 passed as a msg_t
in	*p3*	parameter 3 passed as a msg_t

Returns

The function return value as a msg_t.

Here is the call graph for this function:



6.3.3.5.3.13 chDelegateCallDirect4()

```
msg_t chDelegateCallDirect4 (
              thread_t * tp,
              delegate_fn4_t func,
              msg_t p1,
              msg_t p2,
              msg_t p3,
              msg_t p4)  [inline], [static]
```

Direct call to a function with four parameters.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use chDelegateCallVeneer() instead.

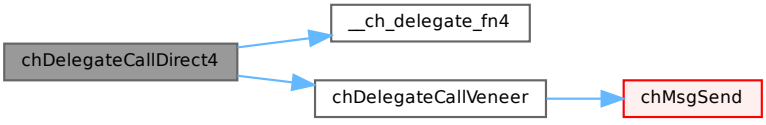
Parameters

in	*tp*	pointer to the delegate thread
in	*func*	pointer to the function to be called
in	*p1*	parameter 1 passed as a msg_t
in	*p2*	parameter 2 passed as a msg_t
in	*p3*	parameter 3 passed as a msg_t
in	*p4*	parameter 4 passed as a msg_t

Returns

The function return value as a msg_t.

Here is the call graph for this function:



6.3.3.6 Jobs Queues

6.3.3.6.1 Detailed Description

Data Structures

- struct ch_jobs_queue
  Type of a jobs queue.
- struct ch_job_descriptor
  Type of a job descriptor.

Macros

- #define MSG_JOB_NULL ((msg_t)-2)
  Dispatcher return code in case of a `JOB_NUL` has been received.

Typedefs

- typedef struct ch_jobs_queue jobs_queue_t
  Type of a jobs queue.
- typedef void(∗ job_function_t) (void ∗arg)
  Type of a job function.
- typedef struct ch_job_descriptor job_descriptor_t
  Type of a job descriptor.

Functions

- static void chJobObjectInit (jobs_queue_t ∗jqp, size_t jobsn, job_descriptor_t ∗jobsbuf, msg_t ∗msgbuf)
  Initializes a jobs queue object.
- static job_descriptor_t ∗ chJobGet (jobs_queue_t ∗jqp)
  Allocates a free job object.
- static job_descriptor_t ∗ chJobGetI (jobs_queue_t ∗jqp)
  Allocates a free job object.
- static job_descriptor_t ∗ chJobGetTimeoutS (jobs_queue_t ∗jqp, sysinterval_t timeout)
  Allocates a free job object.
- static job_descriptor_t ∗ chJobGetTimeout (jobs_queue_t ∗jqp, sysinterval_t timeout)
  Allocates a free job object.
- static void chJobPostI (jobs_queue_t ∗jqp, job_descriptor_t ∗jp)
  Posts a job object.
- static void chJobPostS (jobs_queue_t ∗jqp, job_descriptor_t ∗jp)
  Posts a job object.
- static void chJobPost (jobs_queue_t ∗jqp, job_descriptor_t ∗jp)
  Posts a job object.
- static void chJobPostAheadI (jobs_queue_t ∗jqp, job_descriptor_t ∗jp)
  Posts an high priority job object.
- static void chJobPostAheadS (jobs_queue_t ∗jqp, job_descriptor_t ∗jp)
  Posts an high priority job object.
- static void chJobPostAhead (jobs_queue_t ∗jqp, job_descriptor_t ∗jp)
  Posts an high priority job object.
- static msg_t chJobDispatch (jobs_queue_t ∗jqp)
  Waits for a job then executes it.
- static msg_t chJobDispatchTimeout (jobs_queue_t ∗jqp, sysinterval_t timeout)
  Waits for a job then executes it.

6.3.3.6.2 Macro Definition Documentation

6.3.3.6.2.1 MSG_JOB_NULL

```
#define MSG_JOB_NULL ((msg_t)-2)
```

Dispatcher return code in case of a JOB_NUL has been received.

6.3.3.6.3 Typedef Documentation

6.3.3.6.3.1 jobs_queue_t

```
typedef struct ch_jobs_queue jobs_queue_t
```

Type of a jobs queue.

6.3.3.6.3.2 job_function_t

```
typedef void(* job_function_t) (void *arg)
```

Type of a job function.

6.3.3.6.3.3 job_descriptor_t

```
typedef struct ch_job_descriptor job_descriptor_t
```

Type of a job descriptor.

6.3.3.6.4 Function Documentation

6.3.3.6.4.1 chJobObjectInit()

```
void chJobObjectInit (
             jobs_queue_t * jqp,
             size_t jobsn,
             job_descriptor_t * jobsbuf,
             msg_t * msgbuf)  [inline], [static]
```

Initializes a jobs queue object.

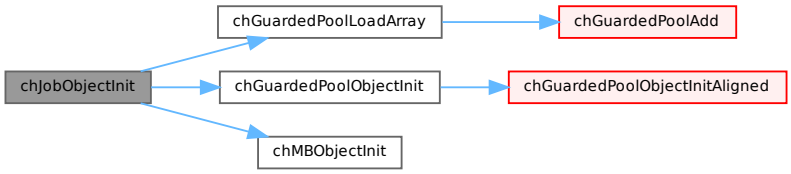
Parameters

out	*jqp*	pointer to a jobs_queue_t structure
in	*jobsn*	number of jobs available
in	*jobsbuf*	pointer to the buffer of jobs, it must be able to hold jobsn job_descriptor_t structures
in	*msgbuf*	pointer to the buffer of messages, it must be able to hold jobsn msg_t messages

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.3.3.6.4.2 chJobGet()

```
job_descriptor_t * chJobGet (
            jobs_queue_t * jqp)  [inline], [static]
```

Allocates a free job object.

Parameters

in	*jqp*	pointer to a jobs_queue_t structure

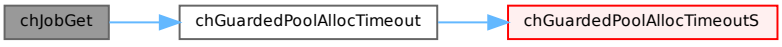
Returns

The pointer to the allocated job object.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.6.4.3 chJobGetI()

```
job_descriptor_t * chJobGetI (
            jobs_queue_t * jqp)  [inline], [static]
```

Allocates a free job object.

Parameters

in	*jqp*	pointer to a jobs_queue_t structure

Returns

The pointer to the allocated job object.

Return values

NULL	if a job object is not immediately available.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.3.3.6.4.4 chJobGetTimeoutS()

```
job_descriptor_t * chJobGetTimeoutS (
            jobs_queue_t * jqp,
            sysinterval_t timeout)  [inline], [static]
```

Allocates a free job object.

Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The pointer to the allocated job object.

Return values

NULL	if a job object is not available within the specified timeout.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

6.3.3.6.4.5 chJobGetTimeout()

```
job_descriptor_t * chJobGetTimeout (
            jobs_queue_t * jqp,
            sysinterval_t timeout)  [inline], [static]
```

Allocates a free job object.

Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The pointer to the allocated job object.

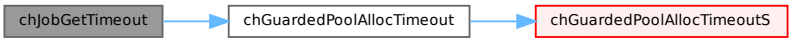
Return values

NULL	if a job object is not available within the specified timeout.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.6.4.6 chJobPostI()

```
void chJobPostI (
            jobs_queue_t * jqp,
            job_descriptor_t * jp)  [inline], [static]
```

Posts a job object.

Note

By design the object can be always immediately posted.

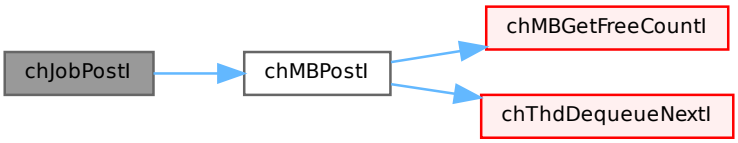
Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*jp*	pointer to the job object to be posted

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.3.3.6.4.7 chJobPostS()

```
void chJobPostS (
            jobs_queue_t * jqp,
            job_descriptor_t * jp)  [inline], [static]
```

Posts a job object.

Note

By design the object can be always immediately posted.

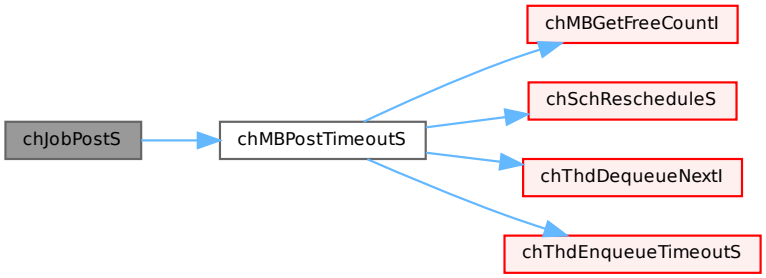
Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*jp*	pointer to the job object to be posted

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.3.6.4.8 chJobPost()

```
void chJobPost (
             jobs_queue_t * jqp,
             job_descriptor_t * jp)  [inline], [static]
```

Posts a job object.

Note

By design the object can be always immediately posted.

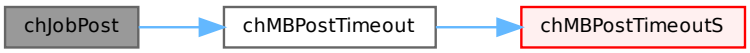
Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*jp*	pointer to the job object to be posted

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.6.4.9 chJobPostAheadI()

```
void chJobPostAheadI (
            jobs_queue_t * jqp,
            job_descriptor_t * jp)  [inline], [static]
```

Posts an high priority job object.

Note

By design the object can be always immediately posted.

Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*jp*	pointer to the job object to be posted

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.3.3.6.4.10 chJobPostAheadS()

```
void chJobPostAheadS (
            jobs_queue_t * jqp,
            job_descriptor_t * jp)  [inline], [static]
```

Posts an high priority job object.

Note

By design the object can be always immediately posted.

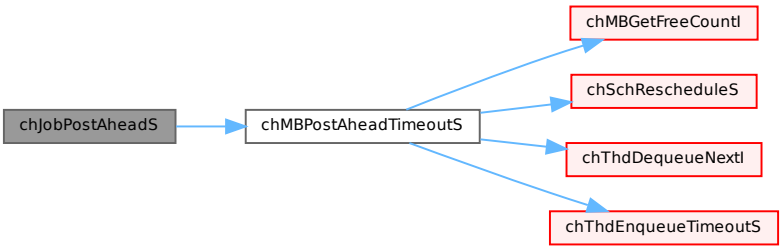
Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*jp*	pointer to the job object to be posted

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.3.6.4.11 chJobPostAhead()

```
void chJobPostAhead (
              jobs_queue_t * jqp,
              job_descriptor_t * jp)  [inline], [static]
```

Posts an high priority job object.

Note

By design the object can be always immediately posted.

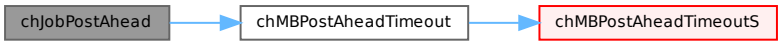
Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*jp*	pointer to the job object to be posted

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.3.6.4.12 chJobDispatch()

```
msg_t chJobDispatch (
              jobs_queue_t * jqp)  [inline], [static]
```

Waits for a job then executes it.

Parameters

in	*jqp*	pointer to a jobs_queue_t structure

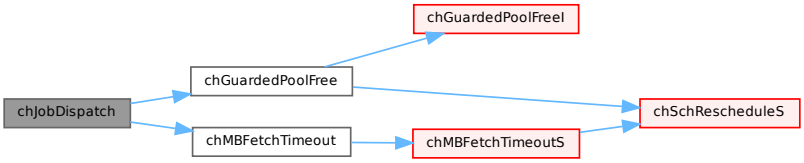
Returns

The function outcome.

Return values

MSG_OK	if a job has been executed.
MSG_RESET	if the internal mailbox has been reset.
MSG_JOB_NULL	if a JOB_NULL has been received.

Here is the call graph for this function:



6.3.3.6.4.13 chJobDispatchTimeout()

```
msg_t chJobDispatchTimeout (
            jobs_queue_t * jqp,
            sysinterval_t timeout)  [inline], [static]
```

Waits for a job then executes it.

Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

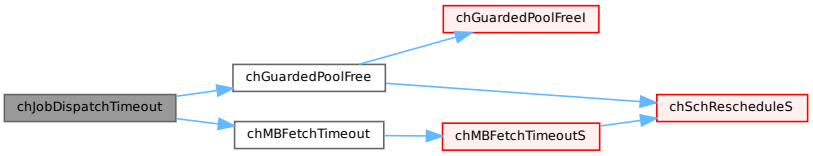
Returns

The function outcome.

Return values

MSG_OK	if a job has been executed.
MSG_TIMEOUT	if a timeout occurred.
MSG_RESET	if the internal mailbox has been reset.
MSG_JOB_NULL	if a `JOB_NULL` has been received.

Here is the call graph for this function:



6.3.4 Memory Management

6.3.4.1 Detailed Description

Memory Management services.

Topics

- Core Memory Manager
- Memory Heaps
- Memory Pools

6.3.4.2 Core Memory Manager

6.3.4.2.1 Detailed Description

Core Memory Manager related APIs and services.

Operation mode

The core memory manager is a simplified allocator that only allows to allocate memory blocks without the possibility to free them.
This allocator is meant as a memory blocks provider for the other allocators such as:

- C-Runtime allocator (through a compiler specific adapter module).
- Heap allocator (see Memory Heaps).
- Memory pools allocator (see Memory Pools).

By having a centralized memory provider the various allocators can coexist and share the main memory.
This allocator, alone, is also useful for very simple applications that just require a simple way to get memory blocks.

Precondition

In order to use the core memory manager APIs the `CH_CFG_USE_MEMCORE` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

Data Structures

- struct memcore_t

  Type of memory core object.

Macros

- #define CH_CFG_MEMCORE_SIZE 0

  Managed RAM size.
- #define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI

  Allocates a memory block.
- #define chCoreAllocAlignedWithOffset chCoreAllocFromTop

  Allocates a memory block.

Typedefs

- typedef void *(* memgetfunc_t) (size_t size, unsigned align)

  Memory get function.
- typedef void *(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)

  Enhanced memory get function.

Functions

- void __core_init (void)

  Low level memory manager initialization.
- void * chCoreAllocFromBaseI (size_t size, unsigned align, size_t offset)

  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTopI (size_t size, unsigned align, size_t offset)

  Allocates a memory block starting from the top address downward.
- void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)

  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)

  Allocates a memory block starting from the top address downward.
- size_t chCoreGetStatusX (void)

  Core memory status.
- static void * chCoreAllocAlignedI (size_t size, unsigned align)

  Allocates a memory block.
- static void * chCoreAllocAligned (size_t size, unsigned align)

  Allocates a memory block.
- static void * chCoreAllocI (size_t size)

  Allocates a memory block.
- static void * chCoreAlloc (size_t size)

  Allocates a memory block.

Variables

- memcore_t ch_memcore

  Memory core descriptor.

6.3.4.2.2 Macro Definition Documentation

6.3.4.2.2.1 CH_CFG_MEMCORE_SIZE

```
#define CH_CFG_MEMCORE_SIZE 0
```

Managed RAM size.

Size of the RAM area to be managed by the OS. If set to zero then the whole available RAM is used. The core memory is made available to the heap allocator and/or can be used directly through the simplified core memory allocator.

Note

In order to let the OS manage the whole RAM the linker script must provide the `__heap_base__` and `__heap_end__` symbols.

Requires `CH_CFG_USE_MEMCORE`.

6.3.4.2.2.2 chCoreAllocAlignedWithOffsetI

```
#define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI
```

Allocates a memory block.

Note

This is a generic form with unspecified allocation position.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.4.2.2.3 chCoreAllocAlignedWithOffset

```
#define chCoreAllocAlignedWithOffset chCoreAllocFromTop
```

Allocates a memory block.

Note

This is a generic form with unspecified allocation position.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.4.2.3 Typedef Documentation

6.3.4.2.3.1 memgetfunc_t

```
typedef void *(* memgetfunc_t) (size_t size, unsigned align)
```

Memory get function.

6.3.4.2.3.2 memgetfunc2_t

```
typedef void *(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)
```

Enhanced memory get function.

6.3.4.2.4 Function Documentation

6.3.4.2.4.1 __core_init()

```
void __core_init (
             void )
```

Low level memory manager initialization.

Function Class:

Not an API, this function is for internal use only.

6.3.4.2.4.2 chCoreAllocFromBaseI()

```
void * chCoreAllocFromBaseI (
             size_t size,
             unsigned align,
             size_t offset)
```

Allocates a memory block starting from the lowest address upward.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

in	*size*	the size of the block to be allocated.
in	*align*	desired memory alignment
in	*offset*	aligned pointer offset

Returns

A pointer to the allocated memory block.

Return values

NULL	allocation failed, core memory exhausted.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.4.2.4.3 chCoreAllocFromTopI()

```
void * chCoreAllocFromTopI (
            size_t size,
            unsigned align,
            size_t offset)
```

Allocates a memory block starting from the top address downward.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

`in`	*size*	the size of the block to be allocated.
`in`	*align*	desired memory alignment
`in`	*offset*	aligned pointer offset

Returns

A pointer to the allocated memory block.

Return values

NULL	allocation failed, core memory exhausted.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.4.2.4.4 chCoreAllocFromBase()

```
void * chCoreAllocFromBase (
            size_t size,
            unsigned align,
            size_t offset)
```

Allocates a memory block starting from the lowest address upward.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

in	*size*	the size of the block to be allocated.
in	*align*	desired memory alignment
in	*offset*	aligned pointer offset

Returns

A pointer to the allocated memory block.

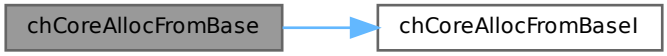
Return values

NULL	allocation failed, core memory exhausted.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.4.2.4.5 chCoreAllocFromTop()

```
void * chCoreAllocFromTop (
            size_t size,
            unsigned align,
            size_t offset)
```

Allocates a memory block starting from the top address downward.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

in	*size*	the size of the block to be allocated.
in	*align*	desired memory alignment
in	*offset*	aligned pointer offset

Returns

A pointer to the allocated memory block.

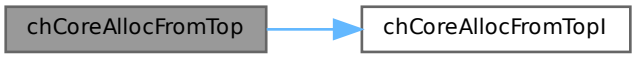
Return values

NULL	allocation failed, core memory exhausted.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.4.2.4.6 chCoreGetStatusX()

```
size_t chCoreGetStatusX (
            void )
```

Core memory status.

Returns

The size, in bytes, of the free core memory.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.3.4.2.4.7 chCoreAllocAlignedI()

```
void * chCoreAllocAlignedI (
            size_t size,
            unsigned align)  [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Note

This is a generic form with unspecified allocation position.

Parameters

in	*size*	the size of the block to be allocated.
in	*align*	desired memory alignment

Returns

A pointer to the allocated memory block.

Return values

NULL	allocation failed, core memory exhausted.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.4.2.4.8 chCoreAllocAligned()

```
void * chCoreAllocAligned (
            size_t size,
            unsigned align)  [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Note

This is a generic form with unspecified allocation position.

Parameters

in	*size*	the size of the block to be allocated
in	*align*	desired memory alignment

Returns

A pointer to the allocated memory block.

Return values

NULL	allocation failed, core memory exhausted.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.4.2.4.9 chCoreAllocI()

```
void * chCoreAllocI (
            size_t size)  [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Note

This is a generic form with unspecified allocation position.

Parameters

in	*size*	the size of the block to be allocated.

Returns

A pointer to the allocated memory block.

Return values

NULL	allocation failed, core memory exhausted.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.4.2.4.10 chCoreAlloc()

```
void * chCoreAlloc (
            size_t size)  [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Note

This is a generic form with unspecified allocation position.

Parameters

in	*size*	the size of the block to be allocated.

Returns

A pointer to the allocated memory block.

Return values

NULL	allocation failed, core memory exhausted.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.4.2.5 Variable Documentation

6.3.4.2.5.1 ch_memcore

```
memcore_t ch_memcore
```

Memory core descriptor.

6.3.4.3 Memory Heaps

6.3.4.3.1 Detailed Description

Heap Allocator related APIs.

Operation mode

The heap allocator implements a first-fit strategy and its APIs are functionally equivalent to the usual `malloc()` and `free()` library functions. The main difference is that the OS heap APIs are guaranteed to be thread safe and there is the ability to return memory blocks aligned to arbitrary powers of two.

Precondition

In order to use the heap APIs the `CH_CFG_USE_HEAP` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

Data Structures

- union heap_header

  Memory heap block header.
- struct memory_heap

  Structure describing a memory heap.

Macros

- #define H_LOCK(h)
- #define H_UNLOCK(h)
- #define H_BLOCK(hp)
- #define H_LIMIT(hp)
- #define H_NEXT(hp)
- #define H_PAGES(hp)
- #define H_HEAP(hp)
- #define H_SIZE(hp)
- #define NPAGES(p1, p2)
- #define CH_HEAP_ALIGNMENT 8U

  Minimum alignment used for heap.
- #define CH_HEAP_AREA(name, size)

  Allocation of an aligned static heap buffer.

Typedefs

- typedef struct memory_heap memory_heap_t

  Type of a memory heap.
- typedef union heap_header heap_header_t

  Type of a memory heap header.

Functions

- void __heap_init (void)

  Initializes the default heap.
- void chHeapObjectInit (memory_heap_t *heapp, void *buf, size_t size)

  Initializes a memory heap from a static memory area.
- void * chHeapAllocAligned (memory_heap_t *heapp, size_t size, unsigned align)

  Allocates a block of memory from the heap by using the first-fit algorithm.
- void chHeapFree (void *p)

  Frees a previously allocated memory block.
- size_t chHeapStatus (memory_heap_t *heapp, size_t *totalp, size_t *largestp)

  Reports the heap status.
- static void * chHeapAlloc (memory_heap_t *heapp, size_t size)

  Allocates a block of memory from the heap by using the first-fit algorithm.
- static size_t chHeapGetSize (const void *p)

  Returns the size of an allocated block.

Variables

- static memory_heap_t default_heap

  Default heap descriptor.

6.3.4.3.2 Macro Definition Documentation

6.3.4.3.2.1 H_LOCK

```
#define H_LOCK(
               h)
```

Value:

```
chMtxLock(&(h)->mtx)
```

6.3.4.3.2.2 H_UNLOCK

```
#define H_UNLOCK(
               h)
```

Value:

```
chMtxUnlock(&(h)->mtx)
```

6.3.4.3.2.3 H_BLOCK

```
#define H_BLOCK(
               hp)
```

Value:

```
((hp) + 1U)
```

6.3.4.3.2.4 H_LIMIT

```
#define H_LIMIT(
               hp)
```

Value:

```
(H_BLOCK(hp) + H_PAGES(hp))
```

6.3.4.3.2.5 H_NEXT

```
#define H_NEXT(
               hp)
```

Value:

```
((hp)->free.next)
```

6.3.4.3.2.6 H_PAGES

```
#define H_PAGES(
               hp)
```

Value:

```
((hp)->free.pages)
```

6.3.4.3.2.7 H_HEAP

```
#define H_HEAP(
               hp)
```

Value:

```
((hp)->used.heap)
```

6.3.4.3.2.8 H_SIZE

```
#define H_SIZE(
               hp)
```

Value:

```
((hp)->used.size)
```

6.3.4.3.2.9 NPAGES

```
#define NPAGES(
               p1,
               p2)
```

Value:

```
  /*lint -save -e9033 [10.8] The cast is safe.*/                          \
  ((size_t)((p1) - (p2)))                                                 \
  /*lint -restore*/
```

6.3.4.3.2.10 CH_HEAP_ALIGNMENT

```
#define CH_HEAP_ALIGNMENT 8U
```

Minimum alignment used for heap.

Note

Cannot use the sizeof operator in this macro.

6.3.4.3.2.11 CH_HEAP_AREA

```
#define CH_HEAP_AREA(
               name,
               size)
```

Value:

```
  ALIGNED_VAR(CH_HEAP_ALIGNMENT)                                          \
  uint8_t name[MEM_ALIGN_NEXT((size), CH_HEAP_ALIGNMENT)]
```

Allocation of an aligned static heap buffer.

6.3.4.3.3 Typedef Documentation

6.3.4.3.3.1 memory_heap_t

```
typedef struct memory_heap memory_heap_t
```

Type of a memory heap.

6.3.4.3.3.2 heap_header_t

```
typedef union heap_header heap_header_t
```

Type of a memory heap header.

6.3.4.3.4 Function Documentation

6.3.4.3.4.1 __heap_init()

```
void __heap_init (
              void )
```

Initializes the default heap.

Function Class:

Not an API, this function is for internal use only.

6.3.4.3.4.2 chHeapObjectInit()

```
void chHeapObjectInit (
              memory_heap_t * heapp,
              void * buf,
              size_t size)
```

Initializes a memory heap from a static memory area.

Note

The heap buffer base and size are adjusted if the passed buffer is not aligned to `CH_HEAP_ALIGNMENT`. This mean that the effective heap size can be less than `size`.

Parameters

`out`	*heapp*	pointer to the memory heap descriptor to be initialized
`in`	*buf*	heap buffer base
`in`	*size*	heap size

Function Class:

Object or module nitializer function.

6.3.4.3.4.3 chHeapAllocAligned()

```
void * chHeapAllocAligned (
             memory_heap_t * heapp,
             size_t size,
             unsigned align)
```

Allocates a block of memory from the heap by using the first-fit algorithm.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Parameters

in	*heapp*	pointer to a heap descriptor or NULL in order to access the default heap.
in	*size*	the size of the block to be allocated. Note that the allocated block may be a bit bigger than the requested size for alignment and fragmentation reasons.
in	*align*	desired memory alignment

Returns

A pointer to the aligned allocated block.

Return values

NULL	if the block cannot be allocated.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.4.3.4.4 chHeapFree()

```
void chHeapFree (
             void * p)
```

Frees a previously allocated memory block.

Parameters

in	*p*	pointer to the memory block to be freed

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.4.3.4.5 chHeapStatus()

```
size_t chHeapStatus (
             memory_heap_t * heapp,
             size_t * totalp,
             size_t * largestp)
```

Reports the heap status.

Note

This function is meant to be used in the test suite, it should not be really useful for the application code.

Parameters

in	*heapp*	pointer to a heap descriptor or NULL in order to access the default heap.
in	*totalp*	pointer to a variable that will receive the total fragmented free space or NULL
in	*largestp*	pointer to a variable that will receive the largest free free block found space or NULL

Returns

The number of fragments in the heap.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.4.3.4.6 chHeapAlloc()

```
void * chHeapAlloc (
            memory_heap_t * heapp,
            size_t size)  [inline], [static]
```

Allocates a block of memory from the heap by using the first-fit algorithm.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Parameters

in	*heapp*	pointer to a heap descriptor or NULL in order to access the default heap.
in	*size*	the size of the block to be allocated. Note that the allocated block may be a bit bigger than the requested size for alignment and fragmentation reasons.

Returns

A pointer to the allocated block.

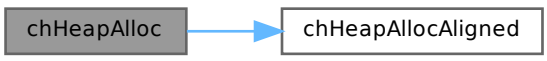
Return values

NULL	if the block cannot be allocated.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.4.3.4.7 chHeapGetSize()

```
size_t chHeapGetSize (
            const void * p)  [inline], [static]
```

Returns the size of an allocated block.

Note

The returned value is the requested size, the real size is the same value aligned to the next CH_HEAP_ALIGNMENT multiple.

Parameters

in	*p*	pointer to the memory block

Returns

Size of the block.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.4.3.5 Variable Documentation

6.3.4.3.5.1 default_heap

```
memory_heap_t default_heap  [static]
```

Default heap descriptor.

6.3.4.4 Memory Pools

6.3.4.4.1 Detailed Description

Memory Pools related APIs and services.

Operation mode

The Memory Pools APIs allow to allocate/free fixed size objects in **constant time** and reliably without memory fragmentation problems.
Memory Pools do not enforce any alignment constraint on the contained object however the objects must be properly aligned to contain a pointer to void.

Precondition

In order to use the memory pools APIs the CH_CFG_USE_MEMPOOLS option must be enabled in chconf.h.

Note

Compatible with RT and NIL.

Data Structures

- struct pool_header

  Memory pool free object header.
- struct memory_pool_t

  Memory pool descriptor.
- struct guarded_memory_pool_t

  Guarded memory pool descriptor.

Macros

- #define __MEMORYPOOL_DATA(name, size, align, provider)

  Data part of a static memory pool initializer.
- #define MEMORYPOOL_DECL(name, size, align, provider)

  Static memory pool initializer.
- #define __GUARDEDMEMORYPOOL_DATA(name, size, align)

  Data part of a static guarded memory pool initializer.
- #define GUARDEDMEMORYPOOL_DECL(name, size, align)

  Static guarded memory pool initializer.

Functions

- void chPoolObjectInitAligned (memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)

  Initializes an empty memory pool.
- void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)

  Loads a memory pool with an array of static objects.
- void * chPoolAllocI (memory_pool_t *mp)

  Allocates an object from a memory pool.
- void * chPoolAlloc (memory_pool_t *mp)

  Allocates an object from a memory pool.
- void chPoolFreeI (memory_pool_t *mp, void *objp)

  Releases an object into a memory pool.
- void chPoolFree (memory_pool_t *mp, void *objp)

  Releases an object into a memory pool.
- void chGuardedPoolObjectInitAligned (guarded_memory_pool_t *gmp, size_t size, unsigned align)

  Initializes an empty guarded memory pool.
- void chGuardedPoolLoadArray (guarded_memory_pool_t *gmp, void *p, size_t n)

  Loads a guarded memory pool with an array of static objects.
- void * chGuardedPoolAllocTimeoutS (guarded_memory_pool_t *gmp, sysinterval_t timeout)

  Allocates an object from a guarded memory pool.
- void * chGuardedPoolAllocTimeout (guarded_memory_pool_t *gmp, sysinterval_t timeout)

  Allocates an object from a guarded memory pool.
- void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)

  Releases an object into a guarded memory pool.
- static void chPoolObjectInit (memory_pool_t *mp, size_t size, memgetfunc_t provider)

  Initializes an empty memory pool.
- static void chPoolAdd (memory_pool_t *mp, void *objp)

  Adds an object to a memory pool.
- static void chPoolAddI (memory_pool_t *mp, void *objp)

  Adds an object to a memory pool.

- static void chGuardedPoolObjectInit (guarded_memory_pool_t *gmp, size_t size)
  Initializes an empty guarded memory pool.
- static cnt_t chGuardedPoolGetCounterI (guarded_memory_pool_t *gmp)
  Gets the count of objects in a guarded memory pool.
- static void * chGuardedPoolAllocI (guarded_memory_pool_t *gmp)
  Allocates an object from a guarded memory pool.
- static void chGuardedPoolFreeI (guarded_memory_pool_t *gmp, void *objp)
  Releases an object into a guarded memory pool.
- static void chGuardedPoolFreeS (guarded_memory_pool_t *gmp, void *objp)
  Releases an object into a guarded memory pool.
- static void chGuardedPoolAdd (guarded_memory_pool_t *gmp, void *objp)
  Adds an object to a guarded memory pool.
- static void chGuardedPoolAddI (guarded_memory_pool_t *gmp, void *objp)
  Adds an object to a guarded memory pool.
- static void chGuardedPoolAddS (guarded_memory_pool_t *gmp, void *objp)
  Adds an object to a guarded memory pool.

6.3.4.4.2 Macro Definition Documentation

6.3.4.4.2.1 __MEMORYPOOL_DATA

```
#define __MEMORYPOOL_DATA(
              name,
              size,
              align,
              provider)
```

Value:

```
{NULL, size, align, provider}
```

Data part of a static memory pool initializer.

This macro should be used when statically initializing a memory pool that is part of a bigger structure.

Parameters

in	*name*	the name of the memory pool variable
in	*size*	size of the memory pool contained objects
in	*align*	required memory alignment
in	*provider*	memory provider function for the memory pool

6.3.4.4.2.2 MEMORYPOOL_DECL

```
#define MEMORYPOOL_DECL(
              name,
              size,
              align,
              provider)
```

Value:

```
memory_pool_t name = __MEMORYPOOL_DATA(name, size, align, provider)
```

Static memory pool initializer.

Statically initialized memory pools require no explicit initialization using `chPoolInit()`.

Parameters

in	*name*	the name of the memory pool variable
in	*size*	size of the memory pool contained objects
in	*align*	required memory alignment
in	*provider*	memory provider function for the memory pool or NULL if the pool is not allowed to grow automatically

6.3.4.4.2.3 __GUARDEDMEMORYPOOL_DATA

```
#define __GUARDEDMEMORYPOOL_DATA(
              name,
              size,
              align)
```

Value:

```
  {                       \
  __SEMAPHORE_DATA(name.sem, (cnt_t)0),                                   \
  __MEMORYPOOL_DATA(NULL, size, align, NULL)                              \
}
```

Data part of a static guarded memory pool initializer.

This macro should be used when statically initializing a memory pool that is part of a bigger structure.

Parameters

in	*name*	the name of the memory pool variable
in	*size*	size of the memory pool contained objects
in	*align*	required memory alignment

6.3.4.4.2.4 GUARDEDMEMORYPOOL_DECL

```
#define GUARDEDMEMORYPOOL_DECL(
              name,
              size,
              align)
```

Value:

```
  guarded_memory_pool_t name = __GUARDEDMEMORYPOOL_DATA(name, size, align)
```

Static guarded memory pool initializer.

Statically initialized guarded memory pools require no explicit initialization using chGuardedPoolInit().

Parameters

in	*name*	the name of the guarded memory pool variable
in	*size*	size of the memory pool contained objects
in	*align*	required memory alignment

6.3.4.4.3 Function Documentation

6.3.4.4.3.1 chPoolObjectInitAligned()

```
void chPoolObjectInitAligned (
             memory_pool_t * mp,
             size_t size,
             unsigned align,
             memgetfunc_t provider)
```

Initializes an empty memory pool.

Parameters

out	*mp*	pointer to a memory_pool_t structure
in	*size*	the size of the objects contained in this memory pool, the minimum accepted size is the size of a pointer to void.
in	*align*	required memory alignment
in	*provider*	memory provider function for the memory pool or NULL if the pool is not allowed to grow automatically

Function Class:

Object or module nitializer function.

6.3.4.4.3.2 chPoolLoadArray()

```
void chPoolLoadArray (
             memory_pool_t * mp,
             void * p,
             size_t n)
```

Loads a memory pool with an array of static objects.

Precondition

The memory pool must already be initialized.

The array elements must be of the right size for the specified memory pool.

The array elements size must be a multiple of the alignment requirement for the pool.

Postcondition

The memory pool contains the elements of the input array.

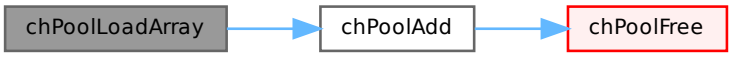
Parameters

in	*mp*	pointer to a memory_pool_t structure
in	*p*	pointer to the array first element
in	*n*	number of elements in the array

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.4.4.3.3 chPoolAllocI()

```
void * chPoolAllocI (
            memory_pool_t * mp)
```

Allocates an object from a memory pool.

Precondition

The memory pool must already be initialized.

Parameters

in	*mp*	pointer to a memory_pool_t structure

Returns

The pointer to the allocated object.

Return values

NULL	if pool is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.4.4.3.4 chPoolAlloc()

```
void * chPoolAlloc (
            memory_pool_t * mp)
```

Allocates an object from a memory pool.

Precondition

The memory pool must already be initialized.

Parameters

in	*mp*	pointer to a memory_pool_t structure

Returns

The pointer to the allocated object.

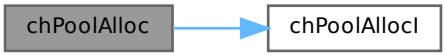
Return values

NULL	if pool is empty.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.4.4.3.5 chPoolFreeI()

```
void chPoolFreeI (
              memory_pool_t * mp,
              void * objp)
```

Releases an object into a memory pool.

Precondition

The memory pool must already be initialized.

The freed object must be of the right size for the specified memory pool.

The added object must be properly aligned.

Parameters

in	*mp*	pointer to a memory_pool_t structure
in	*objp*	the pointer to the object to be released

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.4.4.3.6 chPoolFree()

```
void chPoolFree (
            memory_pool_t * mp,
            void * objp)
```

Releases an object into a memory pool.

Precondition

The memory pool must already be initialized.

The freed object must be of the right size for the specified memory pool.

The added object must be properly aligned.

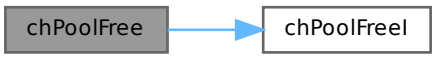
Parameters

in	*mp*	pointer to a memory_pool_t structure
in	*objp*	the pointer to the object to be released

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.4.4.3.7 chGuardedPoolObjectInitAligned()

```
void chGuardedPoolObjectInitAligned (
            guarded_memory_pool_t * gmp,
            size_t size,
            unsigned align)
```

Initializes an empty guarded memory pool.

Parameters

out	*gmp*	pointer to a guarded_memory_pool_t structure
in	*size*	the size of the objects contained in this guarded memory pool, the minimum accepted size is the size of a pointer to void.
in	*align*	required memory alignment

Function Class:

Object or module nitializer function.

Here is the call graph for this function:

6.3.4.4.3.8 chGuardedPoolLoadArray()

```
void chGuardedPoolLoadArray (
              guarded_memory_pool_t * gmp,
              void * p,
              size_t n)
```

Loads a guarded memory pool with an array of static objects.

Precondition

The guarded memory pool must already be initialized.

The array elements must be of the right size for the specified guarded memory pool.

Postcondition

The guarded memory pool contains the elements of the input array.

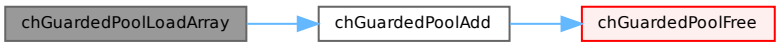
Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*p*	pointer to the array first element
in	*n*	number of elements in the array

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.4.4.3.9 chGuardedPoolAllocTimeoutS()

```
void * chGuardedPoolAllocTimeoutS (
            guarded_memory_pool_t * gmp,
            sysinterval_t timeout)
```

Allocates an object from a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.

Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The pointer to the allocated object.

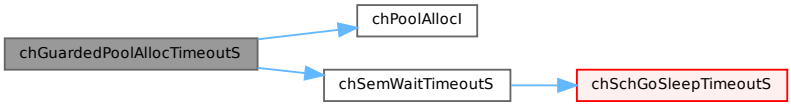
Return values

NULL	if the operation timed out.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.4.4.3.10 chGuardedPoolAllocTimeout()

```
void * chGuardedPoolAllocTimeout (
            guarded_memory_pool_t * gmp,
            sysinterval_t timeout)
```

Allocates an object from a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.

Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The pointer to the allocated object.

Return values

NULL	if the operation timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.4.4.3.11 chGuardedPoolFree()

```
void chGuardedPoolFree (
              guarded_memory_pool_t * gmp,
              void * objp)
```

Releases an object into a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.

The freed object must be of the right size for the specified guarded memory pool.

The added object must be properly aligned.

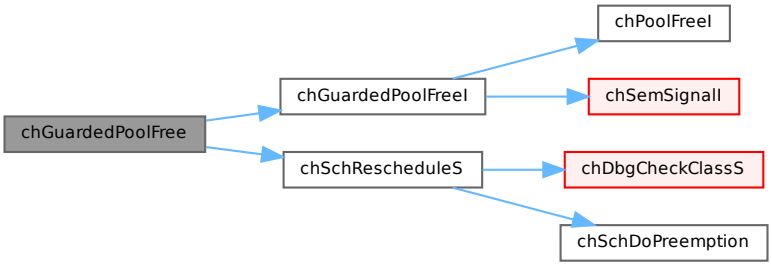
Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*objp*	the pointer to the object to be released

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.4.4.3.12 chPoolObjectInit()

```
void chPoolObjectInit (
              memory_pool_t * mp,
              size_t size,
              memgetfunc_t provider)  [inline], [static]
```

Initializes an empty memory pool.

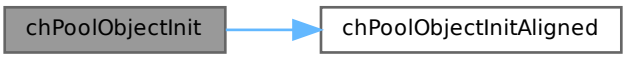
Parameters

out	*mp*	pointer to a memory_pool_t structure
in	*size*	the size of the objects contained in this memory pool, the minimum accepted size is the size of a pointer to void.
in	*provider*	memory provider function for the memory pool or NULL if the pool is not allowed to grow automatically

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.3.4.4.3.13 chPoolAdd()

```
void chPoolAdd (
            memory_pool_t * mp,
            void * objp)  [inline], [static]
```

Adds an object to a memory pool.

Precondition

The memory pool must be already been initialized.

The added object must be of the right size for the specified memory pool.

The added object must be properly aligned.

Note

This function is just an alias for `chPoolFree()` and has been added for clarity.

Parameters

in	*mp*	pointer to a `memory_pool_t` structure
in	*objp*	the pointer to the object to be added

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.4.4.3.14 chPoolAddI()

```
void chPoolAddI (
            memory_pool_t * mp,
            void * objp)  [inline], [static]
```

Adds an object to a memory pool.

Precondition

The memory pool must be already been initialized.

The added object must be of the right size for the specified memory pool.

The added object must be properly aligned.

Note

This function is just an alias for `chPoolFreeI()` and has been added for clarity.

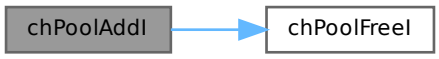
Parameters

in	*mp*	pointer to a memory_pool_t structure
in	*objp*	the pointer to the object to be added

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.3.4.4.3.15 chGuardedPoolObjectInit()

```
void chGuardedPoolObjectInit (
            guarded_memory_pool_t * gmp,
            size_t size)  [inline], [static]
```

Initializes an empty guarded memory pool.

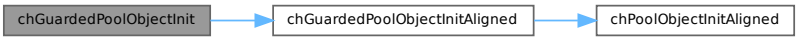
Parameters

out	*gmp*	pointer to a guarded_memory_pool_t structure
in	*size*	the size of the objects contained in this guarded memory pool, the minimum accepted size is the size of a pointer to void.

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.3.4.4.3.16 chGuardedPoolGetCounterI()

```
cnt_t chGuardedPoolGetCounterI (
            guarded_memory_pool_t * gmp)  [inline], [static]
```

Gets the count of objects in a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure

Returns

The counter of the guard semaphore.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.4.4.3.17 chGuardedPoolAllocI()

```
void * chGuardedPoolAllocI (
            guarded_memory_pool_t * gmp)  [inline], [static]
```

Allocates an object from a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure

Returns

The pointer to the allocated object.

Return values

NULL	if the pool is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.3.4.4.3.18 chGuardedPoolFreeI()

```
void chGuardedPoolFreeI (
            guarded_memory_pool_t * gmp,
            void * objp)  [inline], [static]
```

Releases an object into a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.

The freed object must be of the right size for the specified guarded memory pool.

The added object must be properly aligned.

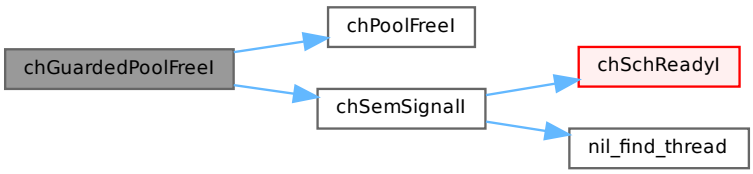
Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*objp*	the pointer to the object to be released

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.3.4.4.3.19 chGuardedPoolFreeS()

```
void chGuardedPoolFreeS (
            guarded_memory_pool_t * gmp,
            void * objp)  [inline], [static]
```

Releases an object into a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.

The freed object must be of the right size for the specified guarded memory pool.

The added object must be properly aligned.

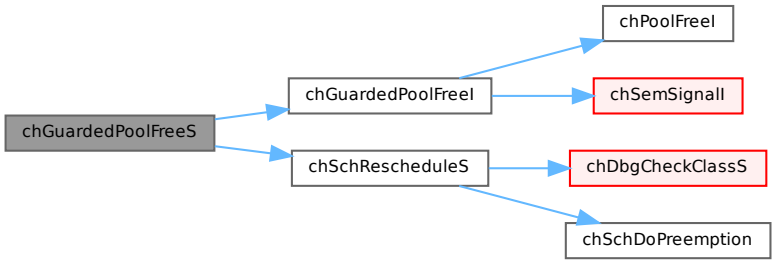
Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*objp*	the pointer to the object to be released

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.4.4.3.20 chGuardedPoolAdd()

```
void chGuardedPoolAdd (
            guarded_memory_pool_t * gmp,
            void * objp)  [inline], [static]
```

Adds an object to a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

The added object must be of the right size for the specified guarded memory pool.

The added object must be properly aligned.

Note

This function is just an alias for chGuardedPoolFree() and has been added for clarity.

Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*objp*	the pointer to the object to be added

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.4.4.3.21 chGuardedPoolAddI()

```
void chGuardedPoolAddI (
            guarded_memory_pool_t * gmp,
            void * objp)  [inline], [static]
```

Adds an object to a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

The added object must be of the right size for the specified guarded memory pool.

The added object must be properly aligned.

Note

This function is just an alias for chGuardedPoolFreeI() and has been added for clarity.

Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*objp*	the pointer to the object to be added

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.3.4.4.3.22 chGuardedPoolAddS()

```
void chGuardedPoolAddS (
            guarded_memory_pool_t * gmp,
            void * objp)  [inline], [static]
```

Adds an object to a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

The added object must be of the right size for the specified guarded memory pool.

The added object must be properly aligned.

Note

This function is just an alias for `chGuardedPoolFreeI()` and has been added for clarity.

Parameters

in	*gmp*	pointer to a `guarded_memory_pool_t` structure
in	*objp*	the pointer to the object to be added

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

6.3.5 Complex Services

6.3.5.1 Detailed Description

Topics

- Objects FIFOs
- Objects Caches
- Dynamic Objects Factory

6.3.5.2 Objects FIFOs

6.3.5.2.1 Detailed Description

Data Structures

- struct ch_objects_fifo

  Type of an objects FIFO.

Typedefs

- typedef struct ch_objects_fifo objects_fifo_t

  Type of an objects FIFO.

Functions

- static void chFifoObjectInitAligned (objects_fifo_t *ofp, size_t objsize, size_t objn, unsigned objalign, void *objbuf, msg_t *msgbuf)

  Initializes a FIFO object.
- static void chFifoObjectInit (objects_fifo_t *ofp, size_t objsize, size_t objn, void *objbuf, msg_t *msgbuf)

  Initializes a FIFO object.
- static void * chFifoTakeObjectI (objects_fifo_t *ofp)

  Allocates a free object.
- static void * chFifoTakeObjectTimeoutS (objects_fifo_t *ofp, sysinterval_t timeout)

  Allocates a free object.
- static void * chFifoTakeObjectTimeout (objects_fifo_t *ofp, sysinterval_t timeout)

  Allocates a free object.
- static void chFifoReturnObjectI (objects_fifo_t *ofp, void *objp)

  Releases a fetched object.
- static void chFifoReturnObjectS (objects_fifo_t *ofp, void *objp)

  Releases a fetched object.
- static void chFifoReturnObject (objects_fifo_t *ofp, void *objp)

  Releases a fetched object.
- static void chFifoSendObjectI (objects_fifo_t *ofp, void *objp)

  Posts an object.
- static void chFifoSendObjectS (objects_fifo_t *ofp, void *objp)

  Posts an object.
- static void chFifoSendObject (objects_fifo_t *ofp, void *objp)

  Posts an object.

- static void chFifoSendObjectAheadI (objects_fifo_t *ofp, void *objp)

  Posts an high priority object.
- static void chFifoSendObjectAheadS (objects_fifo_t *ofp, void *objp)

  Posts an high priority object.
- static void chFifoSendObjectAhead (objects_fifo_t *ofp, void *objp)

  Posts an high priority object.
- static msg_t chFifoReceiveObjectI (objects_fifo_t *ofp, void **objpp)

  Fetches an object.
- static msg_t chFifoReceiveObjectTimeoutS (objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)

  Fetches an object.
- static msg_t chFifoReceiveObjectTimeout (objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)

  Fetches an object.

6.3.5.2.2 Typedef Documentation

6.3.5.2.2.1 objects_fifo_t

```
typedef struct ch_objects_fifo objects_fifo_t
```

Type of an objects FIFO.

6.3.5.2.3 Function Documentation

6.3.5.2.3.1 chFifoObjectInitAligned()

```
void chFifoObjectInitAligned (
              objects_fifo_t * ofp,
              size_t objsize,
              size_t objn,
              unsigned objalign,
              void * objbuf,
              msg_t * msgbuf)  [inline], [static]
```

Initializes a FIFO object.

Precondition

The messages size must be a multiple of the alignment requirement.

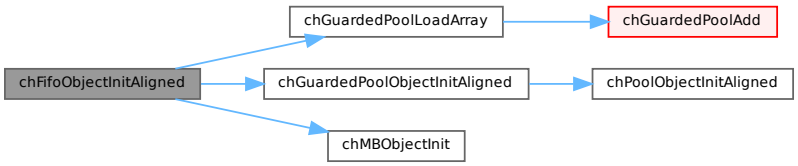
Parameters

out	*ofp*	pointer to a objects_fifo_t structure
in	*objsize*	object size
in	*objn*	number of objects available
in	*objalign*	required objects alignment
in	*objbuf*	pointer to the buffer of objects, it must be able to hold objn objects of objsize size with objalign alignment
in	*msgbuf*	pointer to the buffer of messages, it must be able to hold objn messages

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.3.5.2.3.2 chFifoObjectInit()

```
void chFifoObjectInit (
             objects_fifo_t * ofp,
             size_t objsize,
             size_t objn,
             void * objbuf,
             msg_t * msgbuf)  [inline], [static]
```

Initializes a FIFO object.

Precondition

The messages size must be a multiple of the alignment requirement.

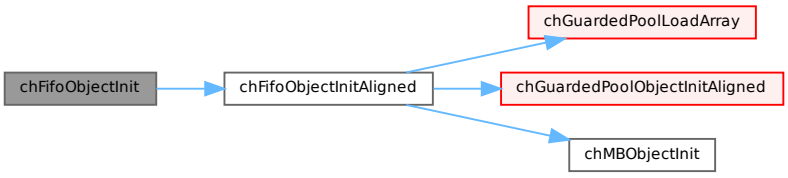
Parameters

out	*ofp*	pointer to a objects_fifo_t structure
in	*objsize*	object size
in	*objn*	number of objects available
in	*objbuf*	pointer to the buffer of objects, it must be able to hold objn objects of objsize size
in	*msgbuf*	pointer to the buffer of messages, it must be able to hold objn messages

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.3.5.2.3.3 chFifoTakeObjectI()

```
void * chFifoTakeObjectI (
            objects_fifo_t * ofp)  [inline], [static]
```

Allocates a free object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure

Returns

The pointer to the allocated object.

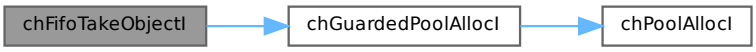
Return values

NULL	if an object is not immediately available.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.3.5.2.3.4 chFifoTakeObjectTimeoutS()

```
void * chFifoTakeObjectTimeoutS (
            objects_fifo_t * ofp,
            sysinterval_t timeout)  [inline], [static]
```

Allocates a free object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The pointer to the allocated object.

Return values

NULL	if an object is not available within the specified timeout.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

6.3.5.2.3.5 chFifoTakeObjectTimeout()

```
void * chFifoTakeObjectTimeout (
            objects_fifo_t * ofp,
            sysinterval_t timeout)  [inline], [static]
```

Allocates a free object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The pointer to the allocated object.

Return values

NULL	if an object is not available within the specified timeout.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.5.2.3.6 chFifoReturnObjectI()

```
void chFifoReturnObjectI (
             objects_fifo_t * ofp,
             void * objp)  [inline], [static]
```

Releases a fetched object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be released

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.3.5.2.3.7 chFifoReturnObjectS()

```
void chFifoReturnObjectS (
             objects_fifo_t * ofp,
             void * objp)  [inline], [static]
```

Releases a fetched object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be released

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

6.3.5.2.3.8 chFifoReturnObject()

```
void chFifoReturnObject (
            objects_fifo_t * ofp,
            void * objp)  [inline], [static]
```

Releases a fetched object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be released

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.5.2.3.9 chFifoSendObjectI()

```
void chFifoSendObjectI (
            objects_fifo_t * ofp,
            void * objp)  [inline], [static]
```

Posts an object.

Note

By design the object can be always immediately posted.

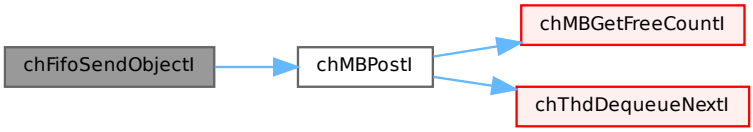
Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be posted

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.3.5.2.3.10 chFifoSendObjectS()

```
void chFifoSendObjectS (
             objects_fifo_t * ofp,
             void * objp)  [inline], [static]
```

Posts an object.

Note

By design the object can be always immediately posted.

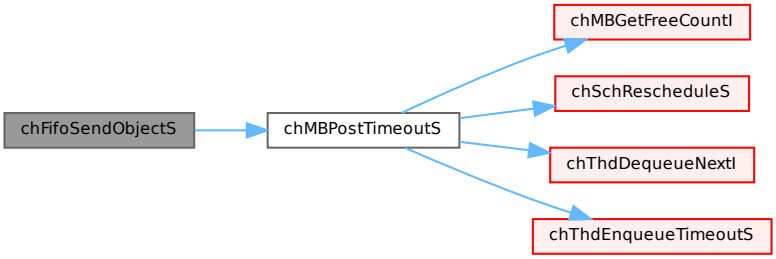
Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be posted

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.5.2.3.11 chFifoSendObject()

```
void chFifoSendObject (
              objects_fifo_t * ofp,
              void * objp)  [inline], [static]
```

Posts an object.

Note

By design the object can be always immediately posted.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be released

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.5.2.3.12 chFifoSendObjectAheadI()

```
void chFifoSendObjectAheadI (
             objects_fifo_t * ofp,
             void * objp)  [inline], [static]
```

Posts an high priority object.

Note

By design the object can be always immediately posted.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be posted

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.3.5.2.3.13 chFifoSendObjectAheadS()

```
void chFifoSendObjectAheadS (
             objects_fifo_t * ofp,
             void * objp)  [inline], [static]
```

Posts an high priority object.

Note

By design the object can be always immediately posted.

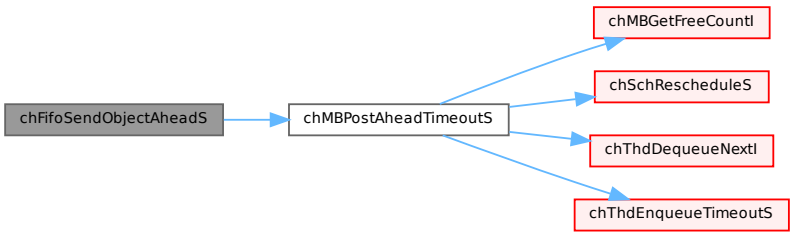
Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be posted

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.5.2.3.14 chFifoSendObjectAhead()

```
void chFifoSendObjectAhead (
            objects_fifo_t * ofp,
            void * objp)  [inline], [static]
```

Posts an high priority object.

Note

By design the object can be always immediately posted.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be released

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.5.2.3.15 chFifoReceiveObjectI()

```
msg_t chFifoReceiveObjectI (
            objects_fifo_t * ofp,
            void ** objpp)  [inline], [static]
```

Fetches an object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objpp*	pointer to the fetched object reference

Returns

The operation status.

Return values

MSG_OK	if an object has been correctly fetched.
MSG_TIMEOUT	if the FIFO is empty and a message cannot be fetched.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.3.5.2.3.16 chFifoReceiveObjectTimeoutS()

```
msg_t chFifoReceiveObjectTimeoutS (
            objects_fifo_t * ofp,
            void ** objpp,
            sysinterval_t timeout)  [inline], [static]
```

Fetches an object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objpp*	pointer to the fetched object reference
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

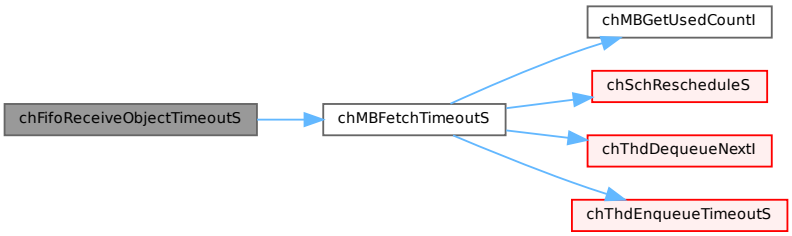
Return values

MSG_OK	if an object has been correctly fetched.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.5.2.3.17 chFifoReceiveObjectTimeout()

```
msg_t chFifoReceiveObjectTimeout (
            objects_fifo_t * ofp,
            void ** objpp,
            sysinterval_t timeout)  [inline], [static]
```

Fetches an object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objpp*	pointer to the fetched object reference
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

Return values

MSG_OK	if an object has been correctly fetched.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.5.3 Objects Caches

6.3.5.3.1 Detailed Description

Cached objects flags

- #define OC_FLAG_INLRU 0x00000001U
- #define OC_FLAG_INHASH 0x00000002U
- #define OC_FLAG_SHARED 0x00000004U
- #define OC_FLAG_NOTSYNC 0x00000008U
- #define OC_FLAG_LAZYWRITE 0x00000010U
- #define OC_FLAG_FORGET 0x00000020U

Data Structures

- struct ch_oc_hash_header

  Structure representing an hash table element.
- struct ch_oc_lru_header

  Structure representing an hash table element.
- struct ch_oc_object

  Structure representing a cached object.
- struct ch_objects_cache

  Structure representing a cache object.

Macros

- #define OC_HASH_FUNCTION(ocp, group, key)
- #define HASH_INSERT(ocp, objp, group, key)
- #define HASH_REMOVE(objp)
- #define LRU_INSERT_HEAD(ocp, objp)
- #define LRU_INSERT_TAIL(ocp, objp)
- #define LRU_REMOVE(objp)

Typedefs

- typedef uint32_t oc_flags_t

  Flags of cached objects.
- typedef struct ch_oc_hash_header oc_hash_header_t

  Type of an hash element header.
- typedef struct ch_oc_lru_header oc_lru_header_t

  Type of an LRU element header.
- typedef struct ch_oc_object oc_object_t

  Type of a cached object.
- typedef struct ch_objects_cache objects_cache_t

  Type of a cache object.
- typedef bool(∗ oc_readf_t) (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)

  Object read function.
- typedef bool(∗ oc_writef_t) (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)

  Object write function.

Functions

- static oc_object_t ∗ hash_get_s (objects_cache_t ∗ocp, uint32_t group, uint32_t key)

  Returns an object pointer from the cache, if present.
- static oc_object_t ∗ lru_get_last_s (objects_cache_t ∗ocp)

  Gets the least recently used object buffer from the LRU list.
- void chCacheObjectInit (objects_cache_t ∗ocp, ucnt_t hashn, oc_hash_header_t ∗hashp, ucnt_t objn, size↩ _t objsz, void ∗objvp, oc_readf_t readf, oc_writef_t writef)

  Initializes a `objects_cache_t` object.
- oc_object_t ∗ chCacheGetObject (objects_cache_t ∗ocp, uint32_t group, uint32_t key)

  Retrieves an object from the cache.
- void chCacheReleaseObjectI (objects_cache_t ∗ocp, oc_object_t ∗objp)

  Releases an object into the cache.
- bool chCacheReadObject (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)

  Reads object data from the storage.
- bool chCacheWriteObject (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)

  Writes the object data back to storage.
- static void chCacheReleaseObject (objects_cache_t ∗ocp, oc_object_t ∗objp)

  Releases an object into the cache.

6.3.5.3.2 Macro Definition Documentation

6.3.5.3.2.1 OC_HASH_FUNCTION

```
#define OC_HASH_FUNCTION(
              ocp,
              group,
              key)
```

Value:

```
(((unsigned)(group) + (unsigned)(key)) & ((unsigned)(ocp)->hashn - 1U))
```

6.3.5.3.2.2 HASH_INSERT

```
#define HASH_INSERT(
                ocp,
                objp,
                group,
                key)
```

Value:

```
  {                                 \
  oc_hash_header_t *hhp;                                                \
  (hhp) = &(ocp)->hashp[OC_HASH_FUNCTION(ocp, group, key)];             \
  (objp)->hash_next = (hhp)->hash_next;                                 \
  (objp)->hash_prev = (oc_object_t *)(hhp);                             \
  (hhp)->hash_next->hash_prev = (objp);                                 \
  (hhp)->hash_next = (objp);                                            \
}
```

6.3.5.3.2.3 HASH_REMOVE

```
#define HASH_REMOVE(
                objp)
```

Value:

```
  {                                                 \
  (objp)->hash_prev->hash_next = (objp)->hash_next;                     \
  (objp)->hash_next->hash_prev = (objp)->hash_prev;                     \
}
```

6.3.5.3.2.4 LRU_INSERT_HEAD

```
#define LRU_INSERT_HEAD(
                ocp,
                objp)
```

Value:

```
  {                                     \
  (objp)->lru_next = (ocp)->lru.lru_next;                               \
  (objp)->lru_prev = (oc_object_t *)&(ocp)->lru;                        \
  (ocp)->lru.lru_next->lru_prev = (objp);                               \
  (ocp)->lru.lru_next = (objp);                                         \
}
```

6.3.5.3.2.5 LRU_INSERT_TAIL

```
#define LRU_INSERT_TAIL(
                ocp,
                objp)
```

Value:

```
  {                                     \
  (objp)->lru_prev = (ocp)->lru.lru_prev;                               \
  (objp)->lru_next = (oc_object_t *)&(ocp)->lru;                        \
  (ocp)->lru.lru_prev->lru_next = (objp);                               \
  (ocp)->lru.lru_prev = (objp);                                         \
}
```

6.3.5.3.2.6 LRU_REMOVE

```
#define LRU_REMOVE(
                objp)
```

Value:

```
  {                                                   \
  (objp)->lru_prev->lru_next = (objp)->lru_next;                          \
  (objp)->lru_next->lru_prev = (objp)->lru_prev;                          \
}
```

6.3.5.3.2.7 OC_FLAG_INLRU

```
#define OC_FLAG_INLRU 0x00000001U
```

6.3.5.3.2.8 OC_FLAG_INHASH

```
#define OC_FLAG_INHASH 0x00000002U
```

6.3.5.3.2.9 OC_FLAG_SHARED

```
#define OC_FLAG_SHARED 0x00000004U
```

6.3.5.3.2.10 OC_FLAG_NOTSYNC

```
#define OC_FLAG_NOTSYNC 0x00000008U
```

6.3.5.3.2.11 OC_FLAG_LAZYWRITE

```
#define OC_FLAG_LAZYWRITE 0x00000010U
```

6.3.5.3.2.12 OC_FLAG_FORGET

```
#define OC_FLAG_FORGET 0x00000020U
```

6.3.5.3.3 Typedef Documentation

6.3.5.3.3.1 oc_flags_t

```
typedef uint32_t oc_flags_t
```

Flags of cached objects.

6.3.5.3.3.2 oc_hash_header_t

```
typedef struct ch_oc_hash_header oc_hash_header_t
```

Type of an hash element header.

6.3.5.3.3.3 oc_lru_header_t

```
typedef struct ch_oc_lru_header oc_lru_header_t
```

Type of an LRU element header.

6.3.5.3.3.4 oc_object_t

```
typedef struct ch_oc_object oc_object_t
```

Type of a cached object.

6.3.5.3.3.5 objects_cache_t

```
typedef struct ch_objects_cache objects_cache_t
```

Type of a cache object.

6.3.5.3.3.6 oc_readf_t

```
typedef bool(* oc_readf_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)
```

Object read function.

Parameters

in	*ocp*	pointer to the objects_cache_t structure
in	*async*	requests an asynchronous operation if supported, the function is then responsible for releasing the object

6.3.5.3.3.7 oc_writef_t

```
typedef bool(* oc_writef_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)
```

Object write function.

Parameters

in	*ocp*	pointer to the objects_cache_t structure
in	*async*	requests an asynchronous operation if supported, the function is then responsible for releasing the object

6.3.5.3.4 Function Documentation

6.3.5.3.4.1 hash_get_s()

```
oc_object_t * hash_get_s (
              objects_cache_t * ocp,
              uint32_t group,
              uint32_t key)  [static]
```

Returns an object pointer from the cache, if present.

Parameters

out	*ocp*	pointer to the objects_cache_t structure to be
in	*group*	object group identifier
in	*key*	object identifier within the group initialized

Returns

The pointer to the retrieved object.

Return values

NULL	if the object is not in cache.

Function Class:

Not an API, this function is for internal use only.

6.3.5.3.4.2 lru_get_last_s()

```
oc_object_t * lru_get_last_s (
              objects_cache_t * ocp)  [static]
```

Gets the least recently used object buffer from the LRU list.

Parameters

out	*ocp*	pointer to the objects_cache_t structure to be

Returns

The pointer to the retrieved object.

Function Class:

Not an API, this function is for internal use only.

6.3.5.3.4.3 chCacheObjectInit()

```
void chCacheObjectInit (
              objects_cache_t * ocp,
              ucnt_t hashn,
              oc_hash_header_t * hashp,
              ucnt_t objn,
              size_t objsz,
              void * objvp,
              oc_readf_t readf,
              oc_writef_t writef)
```

Initializes a objects_cache_t object.

Parameters

out	*ocp*	pointer to the objects_cache_t structure to be initialized
in	*hashn*	number of elements in the hash table array, must be a power of two and not lower than objn
in	*hashp*	pointer to the hash table as an array of oc_hash_header_t
in	*objn*	number of elements in the objects table array
in	*objsz*	size of elements in the objects table array, the minimum value is sizeof (oc_object_t).
in	*objvp*	pointer to the hash objects as an array of structures starting with an oc_object_t
in	*readf*	pointer to an object reader function
in	*writef*	pointer to an object writer function

Function Class:

Object or module nitializer function.

6.3.5.3.4.4 chCacheGetObject()

```
oc_object_t * chCacheGetObject (
              objects_cache_t * ocp,
              uint32_t group,
              uint32_t key)
```

Retrieves an object from the cache.

Note

If the object is not in cache then the returned object is marked as OC_FLAG_NOTSYNC meaning that its data contains garbage and must be initialized.

Parameters

in	*ocp*	pointer to the objects_cache_t structure
in	*group*	object group identifier
in	*key*	object identifier within the group

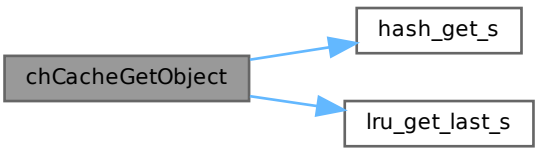
Returns

The pointer to the retrieved object.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.3.4.5 chCacheReleaseObjectI()

```
void chCacheReleaseObjectI (
             objects_cache_t * ocp,
             oc_object_t * objp)
```

Releases an object into the cache.

Note

This function gives a meaning to the following flags:

- OC_FLAG_INLRU must be cleared.
- OC_FLAG_INHASH must be set.
- OC_FLAG_SHARED must be cleared.
- OC_FLAG_NOTSYNC invalidates the object and queues it on the LRU tail.
- OC_FLAG_LAZYWRITE is ignored and kept, a write will occur when the object is removed from the LRU list (lazy write).

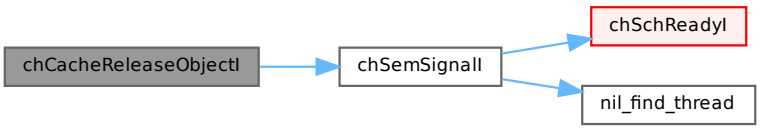
Parameters

in	*ocp*	pointer to the objects_cache_t structure
in	*objp*	pointer to the oc_object_t structure

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.3.5.3.4.6 chCacheReadObject()

```
bool chCacheReadObject (
             objects_cache_t * ocp,
             oc_object_t * objp,
             bool async)
```

Reads object data from the storage.

Note

In case of asynchronous operation an error condition is not reported by this function.

Parameters

in	*ocp*	pointer to the objects_cache_t structure
in	*objp*	pointer to the oc_object_t structure
in	*async*	requests an asynchronous operation if supported, the function is then responsible for releasing the object

Returns

The operation status. In case of asynchronous operation false is always returned.

Return values

false	if the operation succeeded.
true	if the synchronous read operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.5.3.4.7 chCacheWriteObject()

```
bool chCacheWriteObject (
              objects_cache_t * ocp,
              oc_object_t * objp,
              bool async)
```

Writes the object data back to storage.

Note

In case of asynchronous operation an error condition is not reported by this function.

Parameters

in	*ocp*	pointer to the objects_cache_t structure
in	*objp*	pointer to the oc_object_t structure
in	*async*	requests an asynchronous operation if supported, the function is then responsible for releasing the object

Returns

The operation status. In case of asynchronous operation false is always returned.

Return values

false	if the operation succeeded.
true	if the synchronous write operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.5.3.4.8 chCacheReleaseObject()

```
void chCacheReleaseObject (
            objects_cache_t * ocp,
            oc_object_t * objp)  [inline], [static]
```

Releases an object into the cache.

Note

This function gives a meaning to the following flags:

- OC_FLAG_INLRU must be cleared.
- OC_FLAG_INHASH must be set.
- OC_FLAG_SHARED must be cleared.
- OC_FLAG_NOTSYNC invalidates the object and queues it on the LRU tail.
- OC_FLAG_LAZYWRITE is ignored and kept, a write will occur when the object is removed from the LRU list (lazy write).

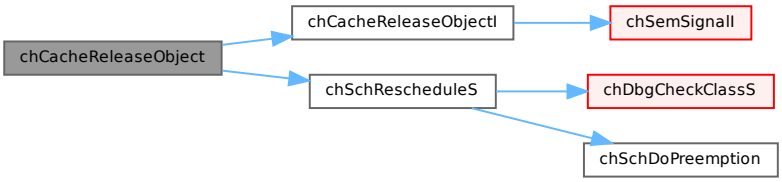
Parameters

in	*ocp*	pointer to the objects_cache_t structure
in	*objp*	pointer to the oc_object_t structure

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4 Dynamic Objects Factory

6.3.5.4.1 Detailed Description

The object factory is a subsystem that allows to:

- Register static objects by name.
- Dynamically create objects and assign them a name.
- Retrieve existing objects by name.
- Free objects by reference.

Allocated OS objects are handled using a reference counter, only when all references have been released then the object memory is freed in a pool.

Precondition

This subsystem requires the `CH_CFG_USE_MEMCORE` and `CH_CFG_USE_MEMPOOLS` options to be set to `TRUE`. The option `CH_CFG_USE_HEAP` is also required if the support for variable length objects is enabled.

Note

Compatible with RT and NIL.

Data Structures

- struct ch_dyn_element

  Type of a dynamic object list element.
- struct ch_dyn_list

  Type of a dynamic object list.
- struct ch_registered_static_object

  Type of a registered object.
- struct ch_dyn_object

  Type of a dynamic buffer object.
- struct ch_dyn_semaphore

  Type of a dynamic semaphore.
- struct ch_dyn_mailbox

  Type of a dynamic buffer object.
- struct ch_dyn_objects_fifo

  Type of a dynamic buffer object.
- struct ch_dyn_pipe

  Type of a dynamic pipe object.
- struct ch_objects_factory

  Type of the factory main object.

Macros

- #define F_LOCK()
- #define F_UNLOCK()
- #define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8

  Maximum length for object names.
- #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

  Enables the registry of generic objects.
- #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

  Enables factory for generic buffers.
- #define CH_CFG_FACTORY_SEMAPHORES TRUE

  Enables factory for semaphores.
- #define CH_CFG_FACTORY_SEMAPHORES FALSE

  Enables factory for semaphores.
- #define CH_CFG_FACTORY_MAILBOXES TRUE

  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_MAILBOXES FALSE

  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

- #define CH_CFG_FACTORY_OBJ_FIFOS FALSE

  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_PIPES TRUE

  Enables factory for Pipes.
- #define CH_CFG_FACTORY_PIPES FALSE

  Enables factory for Pipes.
- #define CH_FACTORY_REQUIRES_POOLS
- #define CH_FACTORY_REQUIRES_HEAP

Typedefs

- typedef struct ch_dyn_element dyn_element_t

  Type of a dynamic object list element.
- typedef struct ch_dyn_list dyn_list_t

  Type of a dynamic object list.
- typedef struct ch_registered_static_object registered_object_t

  Type of a registered object.
- typedef struct ch_dyn_object dyn_buffer_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_semaphore dyn_semaphore_t

  Type of a dynamic semaphore.
- typedef struct ch_dyn_mailbox dyn_mailbox_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_pipe dyn_pipe_t

  Type of a dynamic pipe object.
- typedef struct ch_objects_factory objects_factory_t

  Type of the factory main object.

Functions

- static void copy_name (const char *sp, char *dp)
- static void dyn_list_init (dyn_list_t *dlp)
- static dyn_element_t * dyn_list_find (const char *name, dyn_list_t *dlp)
- static dyn_element_t * dyn_list_find_prev (dyn_element_t *element, dyn_list_t *dlp)
- static dyn_element_t * dyn_list_unlink (dyn_element_t *prev)
- static dyn_element_t * dyn_create_object_heap (const char *name, dyn_list_t *dlp, size_t size, unsigned align)
- static void dyn_release_object_heap (dyn_element_t *dep, dyn_list_t *dlp)
- static dyn_element_t * dyn_create_object_pool (const char *name, dyn_list_t *dlp, memory_pool_t *mp)
- static void dyn_release_object_pool (dyn_element_t *dep, dyn_list_t *dlp, memory_pool_t *mp)
- static dyn_element_t * dyn_find_object (const char *name, dyn_list_t *dlp)
- void __factory_init (void)

  Initializes the objects factory.
- registered_object_t * chFactoryRegisterObject (const char *name, void *objp)

  Registers a generic object.
- registered_object_t * chFactoryFindObject (const char *name)

  Retrieves a registered object.
- registered_object_t * chFactoryFindObjectByPointer (void *objp)

Retrieves a registered object by pointer.

- void chFactoryReleaseObject (registered_object_t *rop)

  Releases a registered object.
- dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)

  Creates a generic dynamic buffer object.
- dyn_buffer_t * chFactoryFindBuffer (const char *name)

  Retrieves a dynamic buffer object.
- void chFactoryReleaseBuffer (dyn_buffer_t *dbp)

  Releases a dynamic buffer object.
- dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)

  Creates a dynamic semaphore object.
- dyn_semaphore_t * chFactoryFindSemaphore (const char *name)

  Retrieves a dynamic semaphore object.
- void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)

  Releases a dynamic semaphore object.
- dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)

  Creates a dynamic mailbox object.
- dyn_mailbox_t * chFactoryFindMailbox (const char *name)

  Retrieves a dynamic mailbox object.
- void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)

  Releases a dynamic mailbox object.
- dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)

  Creates a dynamic "objects FIFO" object.
- dyn_objects_fifo_t * chFactoryFindObjectsFIFO (const char *name)

  Retrieves a dynamic "objects FIFO" object.
- void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)

  Releases a dynamic "objects FIFO" object.
- dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)

  Creates a dynamic pipe object.
- dyn_pipe_t * chFactoryFindPipe (const char *name)

  Retrieves a dynamic pipe object.
- void chFactoryReleasePipe (dyn_pipe_t *dpp)

  Releases a dynamic pipe object.
- static dyn_element_t * chFactoryDuplicateReference (dyn_element_t *dep)

  Duplicates an object reference.
- static void * chFactoryGetObject (registered_object_t *rop)

  Returns the pointer to the inner registered object.
- static size_t chFactoryGetBufferSize (dyn_buffer_t *dbp)

  Returns the size of a generic dynamic buffer object.
- static uint8_t * chFactoryGetBuffer (dyn_buffer_t *dbp)

  Returns the pointer to the inner buffer.
- static semaphore_t * chFactoryGetSemaphore (dyn_semaphore_t *dsp)

  Returns the pointer to the inner semaphore.
- static mailbox_t * chFactoryGetMailbox (dyn_mailbox_t *dmp)

  Returns the pointer to the inner mailbox.
- static objects_fifo_t * chFactoryGetObjectsFIFO (dyn_objects_fifo_t *dofp)

  Returns the pointer to the inner objects FIFO.
- static pipe_t * chFactoryGetPipe (dyn_pipe_t *dpp)

  Returns the pointer to the inner pipe.

Variables

- objects_factory_t ch_factory

  Factory object static instance.

6.3.5.4.2 Macro Definition Documentation

6.3.5.4.2.1 F_LOCK

```
#define F_LOCK()
```

Value:

```
chMtxLock(&ch_factory.mtx)
```

6.3.5.4.2.2 F_UNLOCK

```
#define F_UNLOCK()
```

Value:

```
chMtxUnlock(&ch_factory.mtx)
```

6.3.5.4.2.3 CH_CFG_FACTORY_MAX_NAMES_LENGTH

```
#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8
```

Maximum length for object names.

If the specified length is zero then the name is stored by pointer but this could have unintended side effects.

6.3.5.4.2.4 CH_CFG_FACTORY_OBJECTS_REGISTRY

```
#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE
```

Enables the registry of generic objects.

6.3.5.4.2.5 CH_CFG_FACTORY_GENERIC_BUFFERS

```
#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE
```

Enables factory for generic buffers.

6.3.5.4.2.6 CH_CFG_FACTORY_SEMAPHORES [1/2]

```
#define CH_CFG_FACTORY_SEMAPHORES TRUE
```

Enables factory for semaphores.

6.3.5.4.2.7 CH_CFG_FACTORY_SEMAPHORES [2/2]

```
#define CH_CFG_FACTORY_SEMAPHORES FALSE
```

Enables factory for semaphores.

6.3.5.4.2.8 CH_CFG_FACTORY_MAILBOXES [1/2]

```
#define CH_CFG_FACTORY_MAILBOXES TRUE
```

Enables factory for mailboxes.

6.3.5.4.2.9 CH_CFG_FACTORY_MAILBOXES [2/2]

```
#define CH_CFG_FACTORY_MAILBOXES FALSE
```

Enables factory for mailboxes.

6.3.5.4.2.10 CH_CFG_FACTORY_OBJ_FIFOS [1/3]

```
#define CH_CFG_FACTORY_OBJ_FIFOS TRUE
```

Enables factory for objects FIFOs.

6.3.5.4.2.11 CH_CFG_FACTORY_OBJ_FIFOS [2/3]

```
#define CH_CFG_FACTORY_OBJ_FIFOS TRUE
```

Enables factory for objects FIFOs.

6.3.5.4.2.12 CH_CFG_FACTORY_OBJ_FIFOS [3/3]

```
#define CH_CFG_FACTORY_OBJ_FIFOS FALSE
```

Enables factory for objects FIFOs.

6.3.5.4.2.13 CH_CFG_FACTORY_PIPES [1/2]

```
#define CH_CFG_FACTORY_PIPES TRUE
```

Enables factory for Pipes.

6.3.5.4.2.14 CH_CFG_FACTORY_PIPES [2/2]

```
#define CH_CFG_FACTORY_PIPES FALSE
```

Enables factory for Pipes.

6.3.5.4.2.15 CH_FACTORY_REQUIRES_POOLS

```
#define CH_FACTORY_REQUIRES_POOLS
```

Value:

```
  ((CH_CFG_FACTORY_OBJECTS_REGISTRY == TRUE) ||                         \
   (CH_CFG_FACTORY_SEMAPHORES == TRUE))
```

6.3.5.4.2.16 CH_FACTORY_REQUIRES_HEAP

```
#define CH_FACTORY_REQUIRES_HEAP
```

Value:

```
  ((CH_CFG_FACTORY_GENERIC_BUFFERS == TRUE) ||                          \
   (CH_CFG_FACTORY_MAILBOXES == TRUE) ||                                \
   (CH_CFG_FACTORY_OBJ_FIFOS == TRUE) ||                                \
   (CH_CFG_FACTORY_PIPES == TRUE))
```

6.3.5.4.3 Typedef Documentation

6.3.5.4.3.1 dyn_element_t

```
typedef struct ch_dyn_element dyn_element_t
```

Type of a dynamic object list element.

6.3.5.4.3.2 dyn_list_t

```
typedef struct ch_dyn_list dyn_list_t
```

Type of a dynamic object list.

6.3.5.4.3.3 registered_object_t

```
typedef struct ch_registered_static_object registered_object_t
```

Type of a registered object.

6.3.5.4.3.4 dyn_buffer_t

```
typedef struct ch_dyn_object dyn_buffer_t
```

Type of a dynamic buffer object.

6.3.5.4.3.5 dyn_semaphore_t

```
typedef struct ch_dyn_semaphore dyn_semaphore_t
```

Type of a dynamic semaphore.

6.3.5.4.3.6 dyn_mailbox_t

```
typedef struct ch_dyn_mailbox dyn_mailbox_t
```

Type of a dynamic buffer object.

6.3.5.4.3.7 dyn_objects_fifo_t

```
typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t
```

Type of a dynamic buffer object.

6.3.5.4.3.8 dyn_pipe_t

```
typedef struct ch_dyn_pipe dyn_pipe_t
```

Type of a dynamic pipe object.

6.3.5.4.3.9 objects_factory_t

```
typedef struct ch_objects_factory objects_factory_t
```

Type of the factory main object.

6.3.5.4.4 Function Documentation

6.3.5.4.4.1 copy_name()

```
void copy_name (
            const char * sp,
            char * dp)  [static]
```

6.3.5.4.4.2 dyn_list_init()

```
void dyn_list_init (
            dyn_list_t * dlp)  [inline], [static]
```

6.3.5.4.4.3 dyn_list_find()

```
dyn_element_t * dyn_list_find (
            const char * name,
            dyn_list_t * dlp)  [static]
```

6.3.5.4.4.4 dyn_list_find_prev()

```
dyn_element_t * dyn_list_find_prev (
            dyn_element_t * element,
            dyn_list_t * dlp)  [static]
```

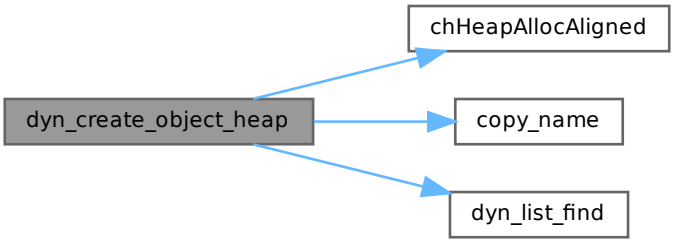
6.3.5.4.4.5 dyn_list_unlink()

```
dyn_element_t * dyn_list_unlink (
            dyn_element_t * prev)  [static]
```

6.3.5.4.4.6 dyn_create_object_heap()

```
dyn_element_t * dyn_create_object_heap (
            const char * name,
            dyn_list_t * dlp,
            size_t size,
            unsigned align)  [static]
```

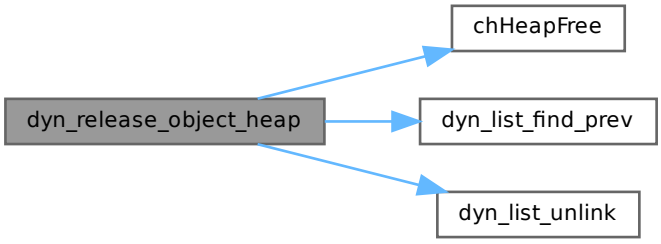
Here is the call graph for this function:



6.3.5.4.4.7 dyn_release_object_heap()

```
void dyn_release_object_heap (
            dyn_element_t * dep,
            dyn_list_t * dlp)  [static]
```

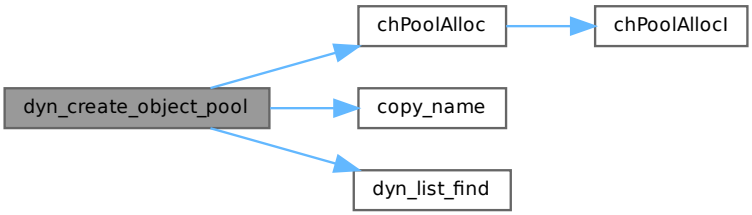
Here is the call graph for this function:



6.3.5.4.4.8 dyn_create_object_pool()

```
dyn_element_t * dyn_create_object_pool (
            const char * name,
            dyn_list_t * dlp,
            memory_pool_t * mp)  [static]
```

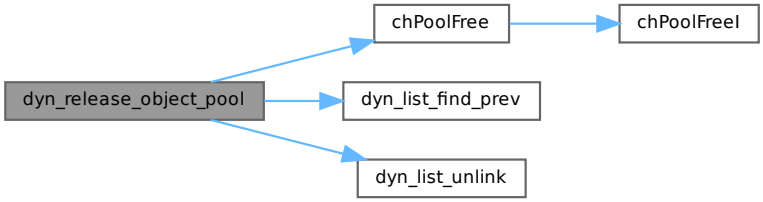
Here is the call graph for this function:



6.3.5.4.4.9 dyn_release_object_pool()

```
void dyn_release_object_pool (
            dyn_element_t * dep,
            dyn_list_t * dlp,
            memory_pool_t * mp)  [static]
```

Here is the call graph for this function:



6.3.5.4.4.10 dyn_find_object()

```
dyn_element_t * dyn_find_object (
            const char * name,
            dyn_list_t * dlp)  [static]
```

Here is the call graph for this function:

6.3.5.4.4.11 __factory_init()

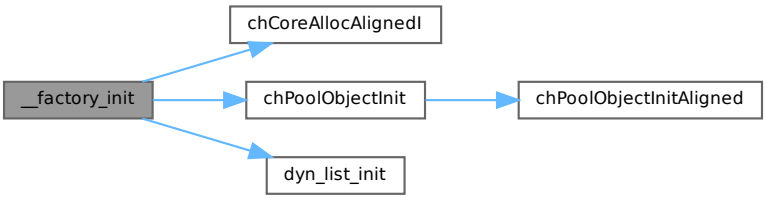
```
void __factory_init (
            void )
```

Initializes the objects factory.

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.3.5.4.4.12 chFactoryRegisterObject()

```
registered_object_t * chFactoryRegisterObject (
            const char * name,
            void * objp)
```

Registers a generic object.

Postcondition

A reference to the registered object is returned and the reference counter is initialized to one.

Parameters

in	*name*	name to be assigned to the registered object
in	*objp*	pointer to the object to be registered

Returns

The reference to the registered object.

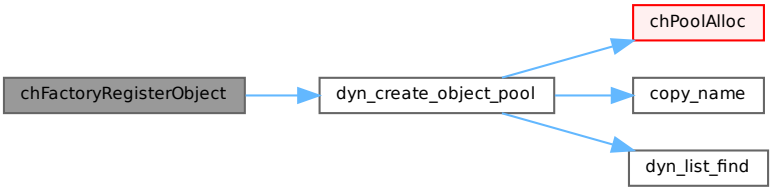
Return values

NULL	if the object to be registered cannot be allocated or a registered object with the same name exists.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4.4.13 chFactoryFindObject()

```
registered_object_t * chFactoryFindObject (
            const char * name)
```

Retrieves a registered object.

Postcondition

A reference to the registered object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the registered object

Returns

The reference to the found registered object.

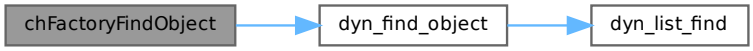
Return values

NULL	if a registered object with the specified name does not exist.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4.4.14 chFactoryFindObjectByPointer()

```
registered_object_t * chFactoryFindObjectByPointer (
            void * objp)
```

Retrieves a registered object by pointer.

Postcondition

A reference to the registered object is returned with the reference counter increased by one.

Parameters

in	*objp*	pointer to the object to be retrieved

Returns

The reference to the found registered object.

Return values

NULL	if a registered object with the specified pointer does not exist.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.5.4.4.15 chFactoryReleaseObject()

```
void chFactoryReleaseObject (
            registered_object_t * rop)
```

Releases a registered object.

The reference counter of the registered object is decreased by one, if reaches zero then the registered object memory is freed.

Note

The object itself is not freed, it could be static, only the allocated list element is freed.

Parameters

in	*rop*	registered object reference

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.5.4.4.16 chFactoryCreateBuffer()

```
dyn_buffer_t * chFactoryCreateBuffer (
            const char * name,
            size_t size)
```

Creates a generic dynamic buffer object.

Postcondition

A reference to the dynamic buffer object is returned and the reference counter is initialized to one.

The dynamic buffer object is filled with zeros.

Parameters

in	*name*	name to be assigned to the new dynamic buffer object
in	*size*	payload size of the dynamic buffer object to be created

Returns

The reference to the created dynamic buffer object.

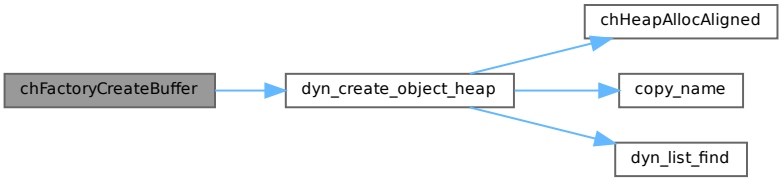
Return values

NULL	if the dynamic buffer object cannot be allocated or a dynamic buffer object with the same name exists.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4.4.17 chFactoryFindBuffer()

```
dyn_buffer_t * chFactoryFindBuffer (
            const char * name)
```

Retrieves a dynamic buffer object.

Postcondition

A reference to the dynamic buffer object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the dynamic buffer object

Returns

The reference to the found dynamic buffer object.

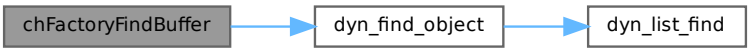
Return values

NULL	if a dynamic buffer object with the specified name does not exist.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4.4.18 chFactoryReleaseBuffer()

```
void chFactoryReleaseBuffer (
             dyn_buffer_t * dbp)
```

Releases a dynamic buffer object.

The reference counter of the dynamic buffer object is decreased by one, if reaches zero then the dynamic buffer object memory is freed.

Parameters

in	*dbp*	dynamic buffer object reference

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.5.4.4.19 chFactoryCreateSemaphore()

```
dyn_semaphore_t * chFactoryCreateSemaphore (
             const char * name,
             cnt_t n)
```

Creates a dynamic semaphore object.

Postcondition

A reference to the dynamic semaphore object is returned and the reference counter is initialized to one.

The dynamic semaphore object is initialized and ready to use.

Parameters

in	*name*	name to be assigned to the new dynamic semaphore object
in	*n*	dynamic semaphore object counter initialization value

Returns

The reference to the created dynamic semaphore object.

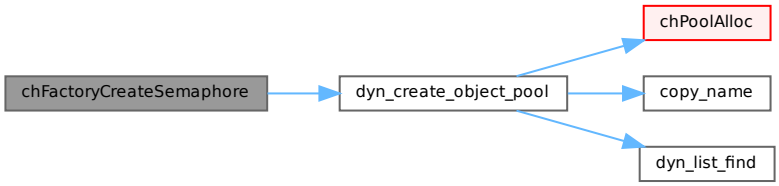
Return values

NULL	if the dynamic semaphore object cannot be allocated or a dynamic semaphore with the same name exists.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4.4.20 chFactoryFindSemaphore()

```
dyn_semaphore_t * chFactoryFindSemaphore (
            const char * name)
```

Retrieves a dynamic semaphore object.

Postcondition

A reference to the dynamic semaphore object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the dynamic semaphore object

Returns

The reference to the found dynamic semaphore object.

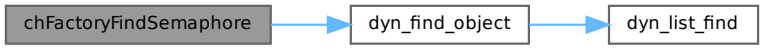
Return values

NULL	if a dynamic semaphore object with the specified name does not exist.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4.4.21 chFactoryReleaseSemaphore()

```
void chFactoryReleaseSemaphore (
            dyn_semaphore_t * dsp)
```

Releases a dynamic semaphore object.

The reference counter of the dynamic semaphore object is decreased by one, if reaches zero then the dynamic semaphore object memory is freed.

Parameters

in	*dsp*	dynamic semaphore object reference

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.5.4.4.22 chFactoryCreateMailbox()

```
dyn_mailbox_t * chFactoryCreateMailbox (
            const char * name,
            size_t n)
```

Creates a dynamic mailbox object.

Postcondition

A reference to the dynamic mailbox object is returned and the reference counter is initialized to one.

The dynamic mailbox object is initialized and ready to use.

Parameters

in	*name*	name to be assigned to the new dynamic mailbox object
in	*n*	mailbox buffer size as number of messages

Returns

The reference to the created dynamic mailbox object.

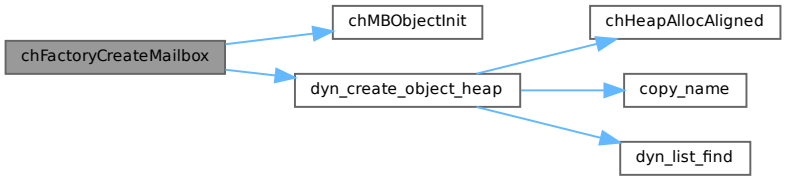
Return values

NULL	if the dynamic mailbox object cannot be allocated or a dynamic mailbox object with the same name exists.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4.4.23 chFactoryFindMailbox()

```
dyn_mailbox_t * chFactoryFindMailbox (
            const char * name)
```

Retrieves a dynamic mailbox object.

Postcondition

A reference to the dynamic mailbox object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the dynamic mailbox object

Returns

The reference to the found dynamic mailbox object.

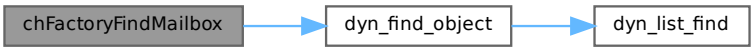
Return values

NULL	if a dynamic mailbox object with the specified name does not exist.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4.4.24 chFactoryReleaseMailbox()

```
void chFactoryReleaseMailbox (
            dyn_mailbox_t * dmp)
```

Releases a dynamic mailbox object.

The reference counter of the dynamic mailbox object is decreased by one, if reaches zero then the dynamic mailbox object memory is freed.

Parameters

in	*dmp*	dynamic mailbox object reference

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.5.4.4.25 chFactoryCreateObjectsFIFO()

```
dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (
            const char * name,
            size_t objsize,
            size_t objn,
            unsigned objalign)
```

Creates a dynamic "objects FIFO" object.

Postcondition

A reference to the dynamic "objects FIFO" object is returned and the reference counter is initialized to one.

The dynamic "objects FIFO" object is initialized and ready to use.

Parameters

in	*name*	name to be assigned to the new dynamic "objects FIFO" object
in	*objsize*	size of objects
in	*objn*	number of objects available
in	*objalign*	required objects alignment

Returns

The reference to the created dynamic "objects FIFO" object.

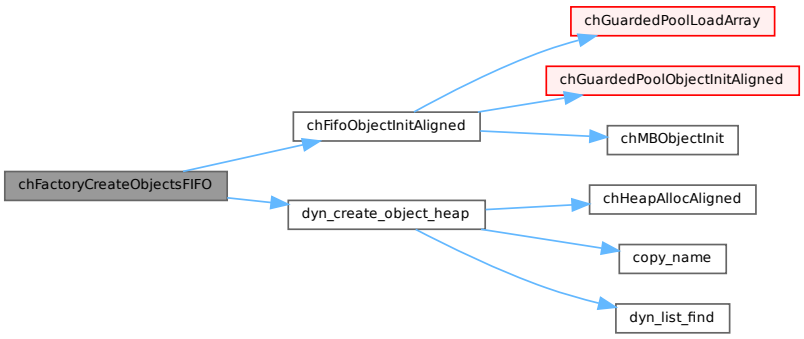
Return values

NULL	if the dynamic "objects FIFO" object cannot be allocated or a dynamic "objects FIFO" object with the same name exists.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4.4.26 chFactoryFindObjectsFIFO()

```
dyn_objects_fifo_t * chFactoryFindObjectsFIFO (
            const char * name)
```

Retrieves a dynamic "objects FIFO" object.

Postcondition

A reference to the dynamic "objects FIFO" object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the dynamic "objects FIFO" object

Returns

The reference to the found dynamic "objects FIFO" object.

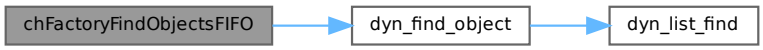
Return values

NULL	if a dynamic "objects FIFO" object with the specified name does not exist.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4.4.27 chFactoryReleaseObjectsFIFO()

```
void chFactoryReleaseObjectsFIFO (
            dyn_objects_fifo_t * dofp)
```

Releases a dynamic "objects FIFO" object.

The reference counter of the dynamic "objects FIFO" object is decreased by one, if reaches zero then the dynamic "objects FIFO" object memory is freed.

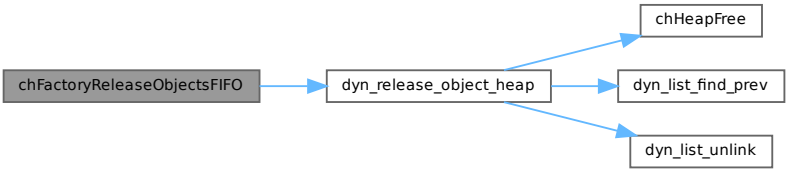
Parameters

in	*dofp*	dynamic "objects FIFO" object reference

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4.4.28 chFactoryCreatePipe()

```
dyn_pipe_t * chFactoryCreatePipe (
             const char * name,
             size_t size)
```

Creates a dynamic pipe object.

Postcondition

A reference to the dynamic pipe object is returned and the reference counter is initialized to one.

The dynamic pipe object is initialized and ready to use.

Parameters

in	*name*	name to be assigned to the new dynamic pipe object
in	*size*	pipe buffer size

Returns

The reference to the created dynamic pipe object.

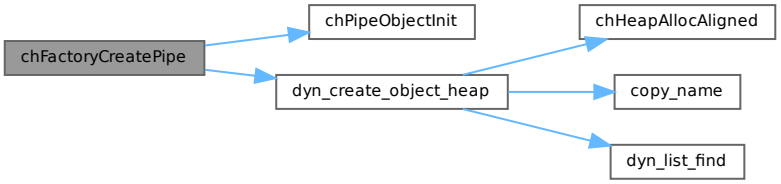
Return values

NULL	if the dynamic pipe object cannot be allocated or a dynamic pipe object with the same name exists.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4.4.29 chFactoryFindPipe()

```
dyn_pipe_t * chFactoryFindPipe (
             const char * name)
```

Retrieves a dynamic pipe object.

Postcondition

A reference to the dynamic pipe object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the pipe object

Returns

The reference to the found dynamic pipe object.

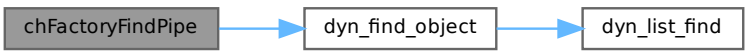
Return values

NULL	if a dynamic pipe object with the specified name does not exist.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.5.4.4.30 chFactoryReleasePipe()

```
void chFactoryReleasePipe (
            dyn_pipe_t * dpp)
```

Releases a dynamic pipe object.

The reference counter of the dynamic pipe object is decreased by one, if reaches zero then the dynamic pipe object memory is freed.

Parameters

in	*dpp*	dynamic pipe object reference

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.5.4.4.31 chFactoryDuplicateReference()

```
dyn_element_t * chFactoryDuplicateReference (
            dyn_element_t * dep)  [inline], [static]
```

Duplicates an object reference.

Note

This function can be used on any kind of dynamic object.

Parameters

in	*dep*	pointer to the element field of the object

Returns

The duplicated object reference.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.5.4.4.32 chFactoryGetObject()

```
void * chFactoryGetObject (
            registered_object_t * rop)  [inline], [static]
```

Returns the pointer to the inner registered object.

Parameters

in	*rop*	registered object reference

Returns

The pointer to the registered object.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.5.4.4.33 chFactoryGetBufferSize()

```
size_t chFactoryGetBufferSize (
            dyn_buffer_t * dbp)  [inline], [static]
```

Returns the size of a generic dynamic buffer object.

Parameters

in	*dbp*	dynamic buffer object reference

Returns

The size of the buffer object in bytes.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.5.4.4.34 chFactoryGetBuffer()

```
uint8_t * chFactoryGetBuffer (
            dyn_buffer_t * dbp)  [inline], [static]
```

Returns the pointer to the inner buffer.

Parameters

in	*dbp*	dynamic buffer object reference

Returns

The pointer to the dynamic buffer.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.5.4.4.35 chFactoryGetSemaphore()

```
semaphore_t * chFactoryGetSemaphore (
              dyn_semaphore_t * dsp)  [inline], [static]
```

Returns the pointer to the inner semaphore.

Parameters

in	*dsp*	dynamic semaphore object reference

Returns

The pointer to the semaphore.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.5.4.4.36 chFactoryGetMailbox()

```
mailbox_t * chFactoryGetMailbox (
              dyn_mailbox_t * dmp)  [inline], [static]
```

Returns the pointer to the inner mailbox.

Parameters

in	*dmp*	dynamic mailbox object reference

Returns

The pointer to the mailbox.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.5.4.4.37 chFactoryGetObjectsFIFO()

```
objects_fifo_t * chFactoryGetObjectsFIFO (
              dyn_objects_fifo_t * dofp)  [inline], [static]
```

Returns the pointer to the inner objects FIFO.

Parameters

in	*dofp*	dynamic "objects FIFO" object reference

Returns

The pointer to the objects FIFO.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.5.4.4.38 chFactoryGetPipe()

```
pipe_t * chFactoryGetPipe (
             dyn_pipe_t * dpp)  [inline], [static]
```

Returns the pointer to the inner pipe.

Parameters

in	*dpp*	dynamic pipe object reference

Returns

The pointer to the pipe.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.5.4.5 Variable Documentation

6.3.5.4.5.1 ch_factory

```
objects_factory_t ch_factory
```

Factory object static instance.

Note

It is a global object because it could be accessed through a specific debugger plugin.

Chapter 7

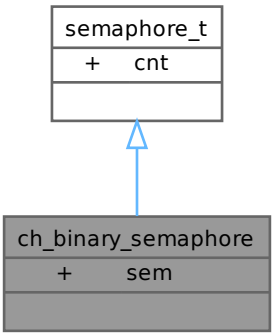
Data Structure Documentation

7.1 ch_binary_semaphore Struct Reference

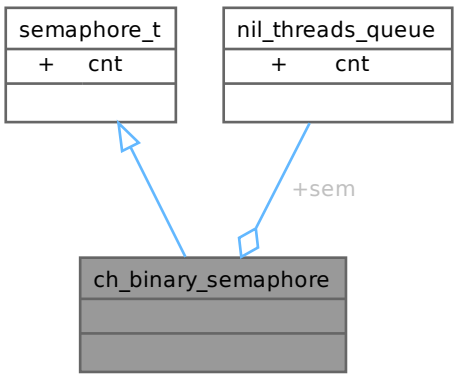
Binary semaphore type.

`#include <chbsem.h>`

Inheritance diagram for ch_binary_semaphore:



Collaboration diagram for ch_binary_semaphore:



Data Fields

- semaphore_t sem

Data Fields inherited from nil_threads_queue

- volatile cnt_t cnt

  Threads Queue counter.

7.1.1 Detailed Description

Binary semaphore type.

7.1.2 Field Documentation

7.1.2.1 sem

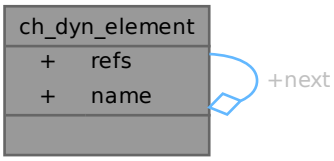
```
semaphore_t ch_binary_semaphore::sem
```

7.2 ch_dyn_element Struct Reference

Type of a dynamic object list element.

`#include <chfactory.h>`

Collaboration diagram for ch_dyn_element:



Data Fields

- struct ch_dyn_element * next
  Next dynamic object in the list.
- ucnt_t refs
  Number of references to this object.
- char name [CH_CFG_FACTORY_MAX_NAMES_LENGTH]

7.2.1 Detailed Description

Type of a dynamic object list element.

7.2.2 Field Documentation

7.2.2.1 next

```
struct ch_dyn_element* ch_dyn_element::next
```

Next dynamic object in the list.

7.2.2.2 refs

```
ucnt_t ch_dyn_element::refs
```

Number of references to this object.

7.2.2.3 name

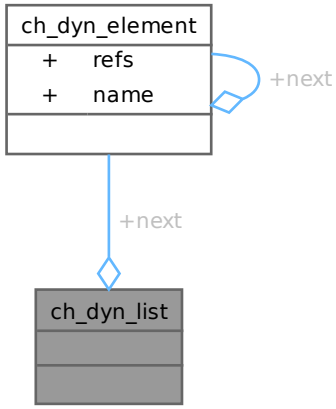
```
char ch_dyn_element::name[CH_CFG_FACTORY_MAX_NAMES_LENGTH]
```

7.3 ch_dyn_list Struct Reference

Type of a dynamic object list.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_list:



Data Fields

- dyn_element_t * next

7.3.1 Detailed Description

Type of a dynamic object list.

7.3.2 Field Documentation

7.3.2.1 next

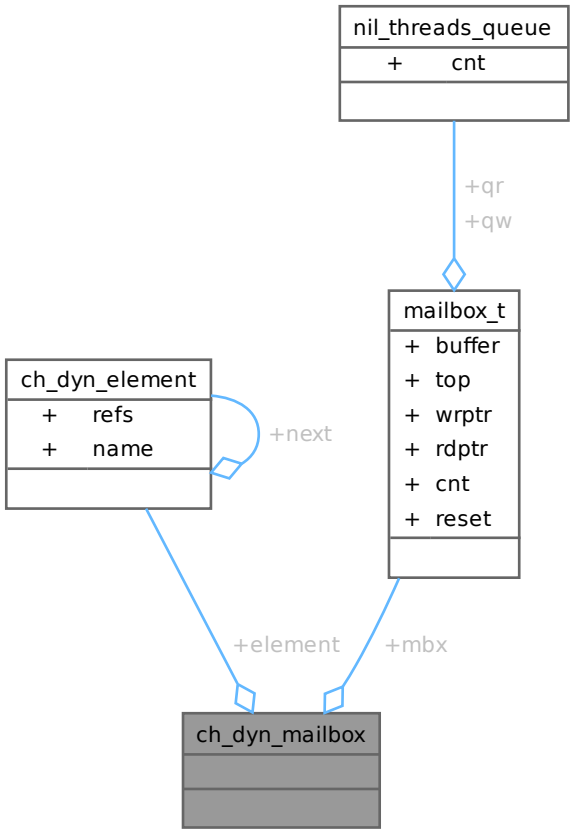
```
dyn_element_t* ch_dyn_list::next
```

7.4 ch_dyn_mailbox Struct Reference

Type of a dynamic buffer object.

`#include <chfactory.h>`

Collaboration diagram for ch_dyn_mailbox:



Data Fields

- dyn_element_t element

  List element of the dynamic buffer object.
- mailbox_t mbx

  The mailbox.

7.4.1 Detailed Description

Type of a dynamic buffer object.

7.4.2 Field Documentation

7.4.2.1 element

```
dyn_element_t ch_dyn_mailbox::element
```

List element of the dynamic buffer object.

7.4.2.2 mbx

```
mailbox_t ch_dyn_mailbox::mbx
```

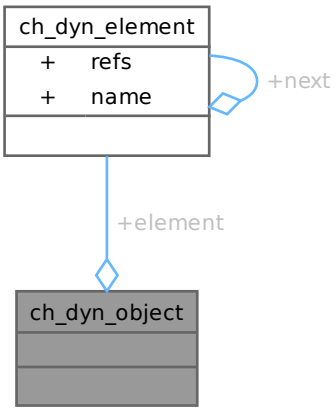
The mailbox.

7.5 ch_dyn_object Struct Reference

Type of a dynamic buffer object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_object:



Data Fields

- dyn_element_t element

  List element of the dynamic buffer object.

7.5.1 Detailed Description

Type of a dynamic buffer object.

7.5.2 Field Documentation

7.5.2.1 element

```
dyn_element_t ch_dyn_object::element
```

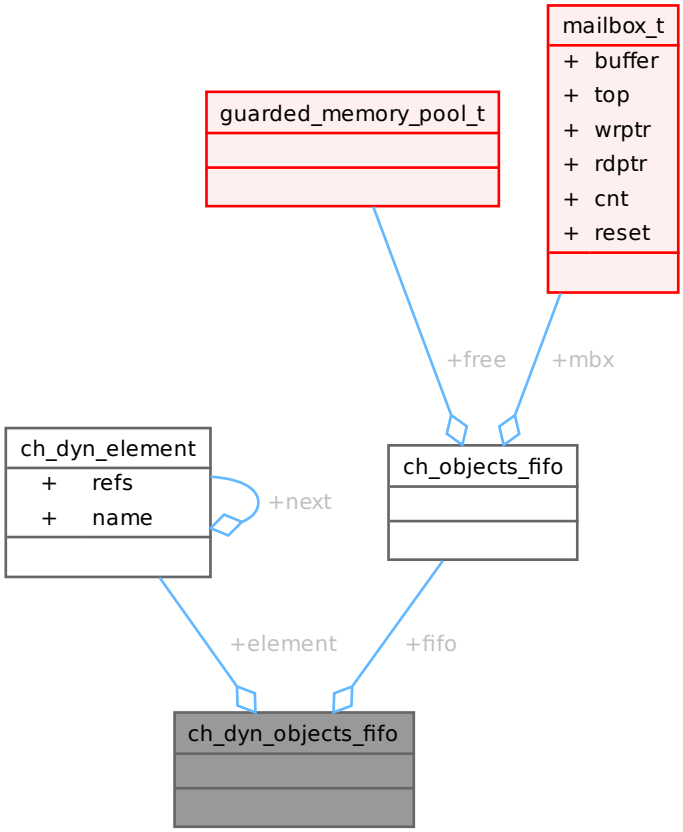
List element of the dynamic buffer object.

7.6 ch_dyn_objects_fifo Struct Reference

Type of a dynamic buffer object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_objects_fifo:



Data Fields

- dyn_element_t element

  List element of the dynamic buffer object.
- objects_fifo_t fifo

  The objects FIFO.

7.6.1 Detailed Description

Type of a dynamic buffer object.

7.6.2 Field Documentation

7.6.2.1 element

```
dyn_element_t ch_dyn_objects_fifo::element
```

List element of the dynamic buffer object.

7.6.2.2 fifo

```
objects_fifo_t ch_dyn_objects_fifo::fifo
```

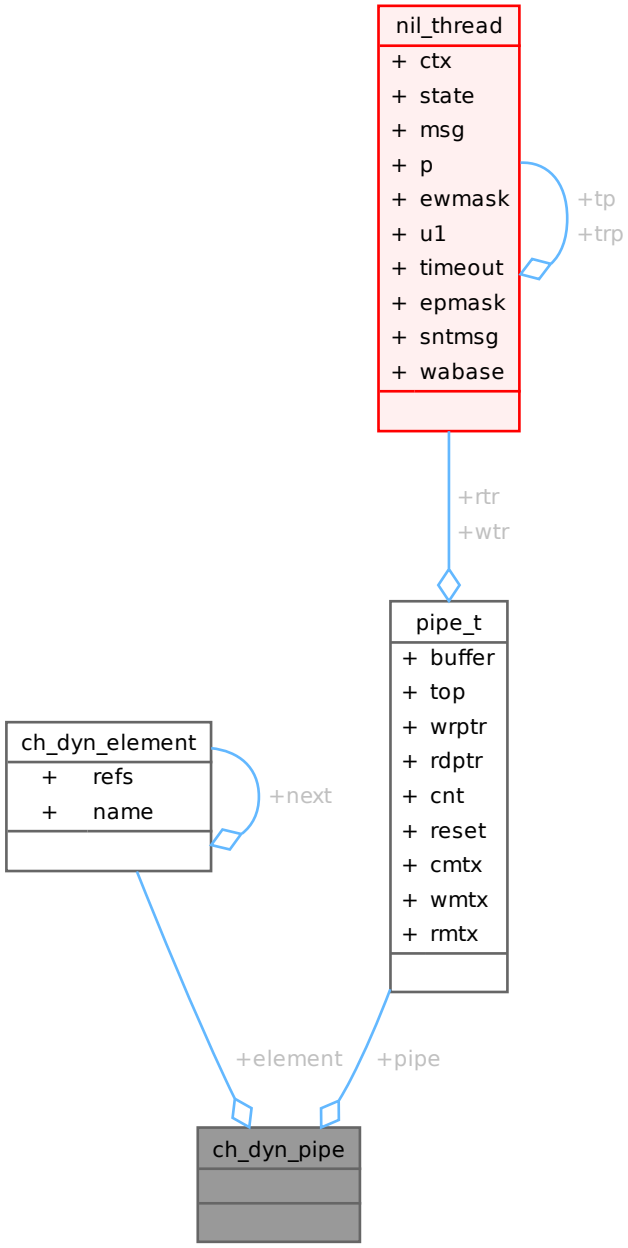
The objects FIFO.

7.7 ch_dyn_pipe Struct Reference

Type of a dynamic pipe object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_pipe:



Data Fields

- dyn_element_t element

  List element of the dynamic pipe object.
- pipe_t pipe

  The pipe.

7.7.1 Detailed Description

Type of a dynamic pipe object.

7.7.2 Field Documentation

7.7.2.1 element

```
dyn_element_t ch_dyn_pipe::element
```

List element of the dynamic pipe object.

7.7.2.2 pipe

```
pipe_t ch_dyn_pipe::pipe
```

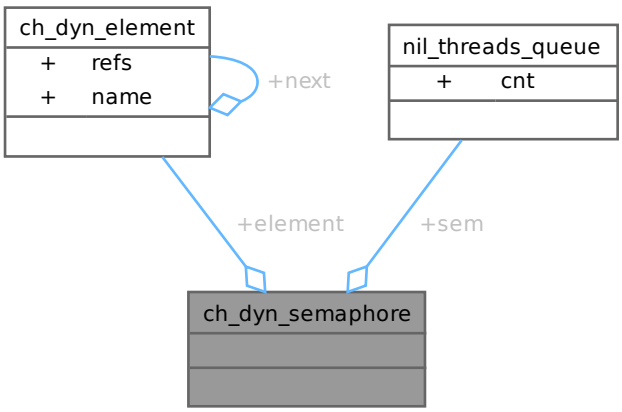
The pipe.

7.8 ch_dyn_semaphore Struct Reference

Type of a dynamic semaphore.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_semaphore:



Data Fields

- dyn_element_t element

  List element of the dynamic semaphore.
- semaphore_t sem

  The semaphore.

7.8.1 Detailed Description

Type of a dynamic semaphore.

7.8.2 Field Documentation

7.8.2.1 element

```
dyn_element_t ch_dyn_semaphore::element
```

List element of the dynamic semaphore.

7.8.2.2 sem

```
semaphore_t ch_dyn_semaphore::sem
```

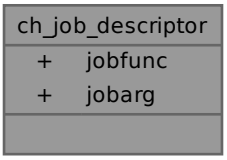
The semaphore.

7.9 ch_job_descriptor Struct Reference

Type of a job descriptor.

```
#include <chjobs.h>
```

Collaboration diagram for ch_job_descriptor:



Data Fields

- job_function_t jobfunc
  Job function.
- void * jobarg
  Argument to be passed to the job function.

7.9.1 Detailed Description

Type of a job descriptor.

7.9.2 Field Documentation

7.9.2.1 jobfunc

```
job_function_t ch_job_descriptor::jobfunc
```

Job function.

7.9.2.2 jobarg

```
void* ch_job_descriptor::jobarg
```

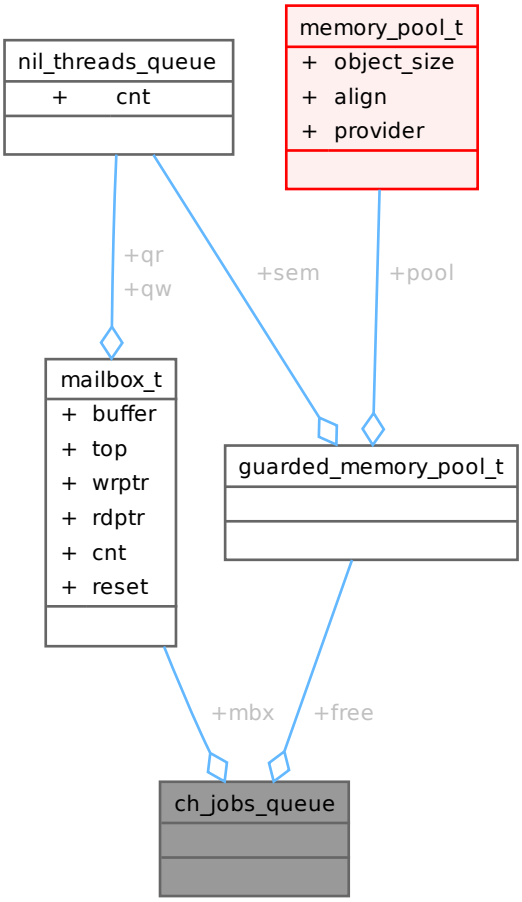
Argument to be passed to the job function.

7.10 ch_jobs_queue Struct Reference

Type of a jobs queue.

```
#include <chjobs.h>
```

Collaboration diagram for ch_jobs_queue:



Data Fields

- guarded_memory_pool_t free
  Pool of the free jobs.
- mailbox_t mbx
  Mailbox of the sent jobs.

7.10.1 Detailed Description

Type of a jobs queue.

7.10.2 Field Documentation

7.10.2.1 free

```
guarded_memory_pool_t ch_jobs_queue::free
```

Pool of the free jobs.

7.10.2.2 mbx

`mailbox_t ch_jobs_queue::mbx`

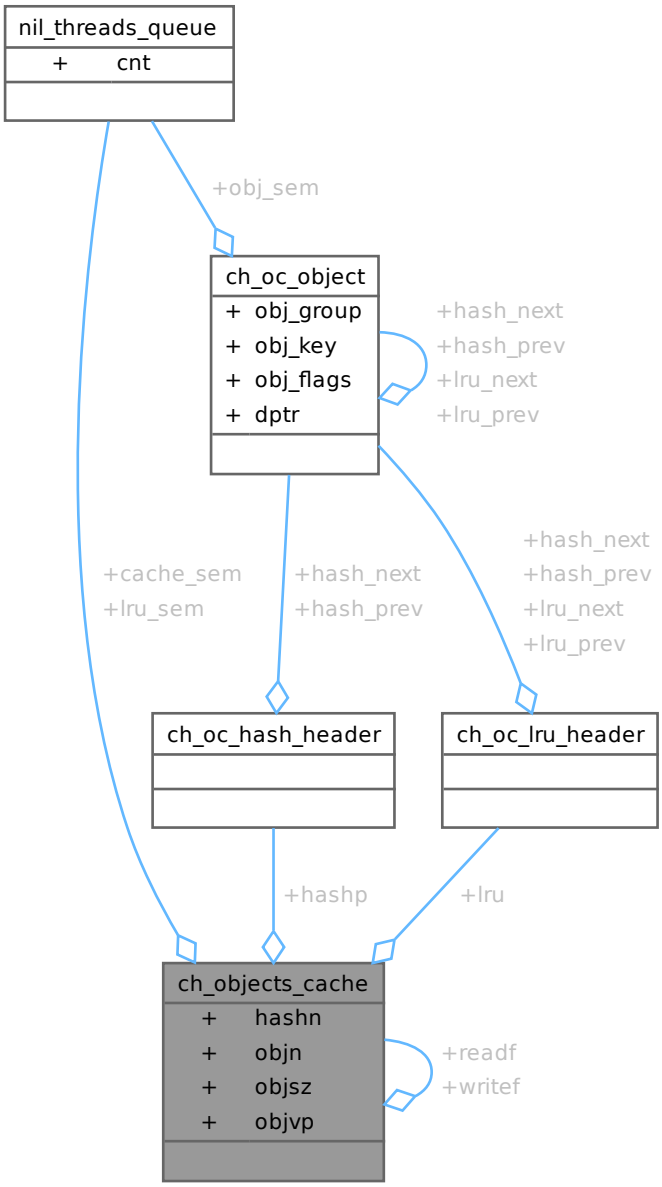
Mailbox of the sent jobs.

7.11 ch_objects_cache Struct Reference

Structure representing a cache object.

`#include <chobjcaches.h>`

Collaboration diagram for ch_objects_cache:



Data Fields

- ucnt_t hashn
  Number of elements in the hash table.
- oc_hash_header_t * hashp
  Pointer to the hash table.
- ucnt_t objn
  Number of elements in the objects table.
- size_t objsz
  Size of elements in the objects table.
- void * objvp
  Pointer to the objects table.
- oc_lru_header_t lru
  LRU list header.
- semaphore_t cache_sem
  Semaphore for cache access.
- semaphore_t lru_sem
  Semaphore for LRU access.
- oc_readf_t readf
  Reader functions for cached objects.
- oc_writef_t writef
  Writer functions for cached objects.

7.11.1 Detailed Description

Structure representing a cache object.

7.11.2 Field Documentation

7.11.2.1 hashn

```
ucnt_t ch_objects_cache::hashn
```

Number of elements in the hash table.

7.11.2.2 hashp

```
oc_hash_header_t* ch_objects_cache::hashp
```

Pointer to the hash table.

7.11.2.3 objn

```
ucnt_t ch_objects_cache::objn
```

Number of elements in the objects table.

7.11.2.4 objsz

```
size_t ch_objects_cache::objsz
```

Size of elements in the objects table.

7.11.2.5 objvp

```
void* ch_objects_cache::objvp
```

Pointer to the objects table.

7.11.2.6 lru

```
oc_lru_header_t ch_objects_cache::lru
```

LRU list header.

7.11.2.7 cache_sem

```
semaphore_t ch_objects_cache::cache_sem
```

Semaphore for cache access.

7.11.2.8 lru_sem

```
semaphore_t ch_objects_cache::lru_sem
```

Semaphore for LRU access.

7.11.2.9 readf

```
oc_readf_t ch_objects_cache::readf
```

Reader functions for cached objects.

7.11.2.10 writef

```
oc_writef_t ch_objects_cache::writef
```

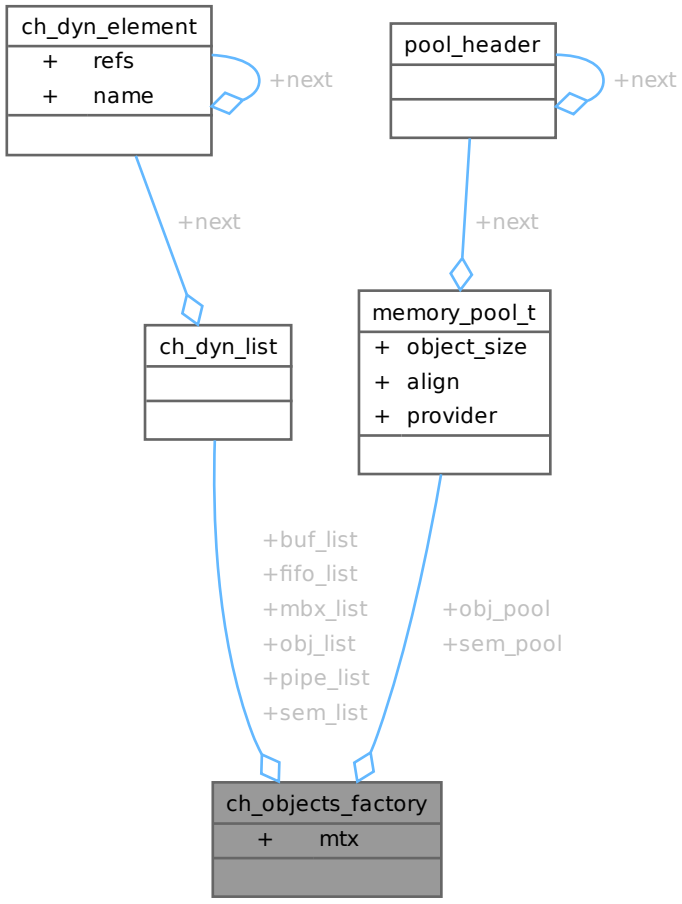
Writer functions for cached objects.

7.12 ch_objects_factory Struct Reference

Type of the factory main object.

`#include <chfactory.h>`

Collaboration diagram for ch_objects_factory:



Data Fields

- mutex_t mtx

  Factory access mutex or semaphore.
- dyn_list_t obj_list

  List of the registered objects.
- memory_pool_t obj_pool

  Pool of the available registered objects.
- dyn_list_t buf_list

  List of the allocated buffer objects.

- dyn_list_t sem_list

  List of the allocated semaphores.
- memory_pool_t sem_pool

  Pool of the available semaphores.
- dyn_list_t mbx_list

  List of the allocated buffer objects.
- dyn_list_t fifo_list

  List of the allocated "objects FIFO" objects.
- dyn_list_t pipe_list

  List of the allocated pipe objects.

7.12.1 Detailed Description

Type of the factory main object.

7.12.2 Field Documentation

7.12.2.1 mtx

```
mutex_t ch_objects_factory::mtx
```

Factory access mutex or semaphore.

7.12.2.2 obj_list

```
dyn_list_t ch_objects_factory::obj_list
```

List of the registered objects.

7.12.2.3 obj_pool

```
memory_pool_t ch_objects_factory::obj_pool
```

Pool of the available registered objects.

7.12.2.4 buf_list

```
dyn_list_t ch_objects_factory::buf_list
```

List of the allocated buffer objects.

7.12.2.5 sem_list

```
dyn_list_t ch_objects_factory::sem_list
```

List of the allocated semaphores.

7.12.2.6 sem_pool

```
memory_pool_t ch_objects_factory::sem_pool
```

Pool of the available semaphores.

7.12.2.7 mbx_list

```
dyn_list_t ch_objects_factory::mbx_list
```

List of the allocated buffer objects.

7.12.2.8 fifo_list

```
dyn_list_t ch_objects_factory::fifo_list
```

List of the allocated "objects FIFO" objects.

7.12.2.9 pipe_list

```
dyn_list_t ch_objects_factory::pipe_list
```

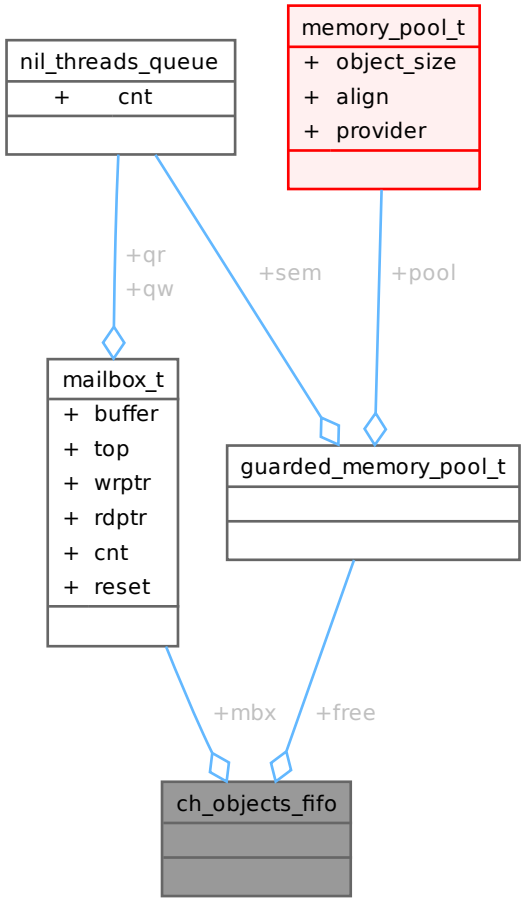
List of the allocated pipe objects.

7.13 ch_objects_fifo Struct Reference

Type of an objects FIFO.

```
#include <chobjfifos.h>
```

Collaboration diagram for ch_objects_fifo:



Data Fields

- guarded_memory_pool_t free
  Pool of the free objects.
- mailbox_t mbx
  Mailbox of the sent objects.

7.13.1 Detailed Description

Type of an objects FIFO.

7.13.2 Field Documentation

7.13.2.1 free

```
guarded_memory_pool_t ch_objects_fifo::free
```

Pool of the free objects.

7.13.2.2 mbx

```
mailbox_t ch_objects_fifo::mbx
```

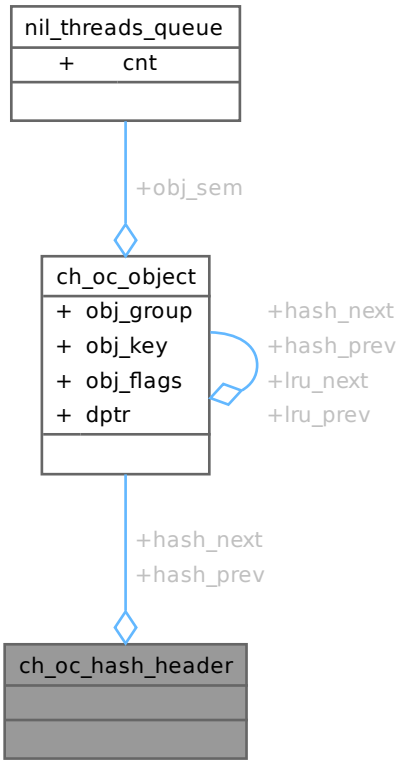
Mailbox of the sent objects.

7.14 ch_oc_hash_header Struct Reference

Structure representing an hash table element.

```
#include <chobjcaches.h>
```

Collaboration diagram for ch_oc_hash_header:



Data Fields

- oc_object_t * hash_next
  Next in the collisions list.
- oc_object_t * hash_prev
  Previous in the collisions list.

7.14.1 Detailed Description

Structure representing an hash table element.

7.14.2 Field Documentation

7.14.2.1 hash_next

```
oc_object_t* ch_oc_hash_header::hash_next
```

Next in the collisions list.

7.14.2.2 hash_prev

```
oc_object_t* ch_oc_hash_header::hash_prev
```

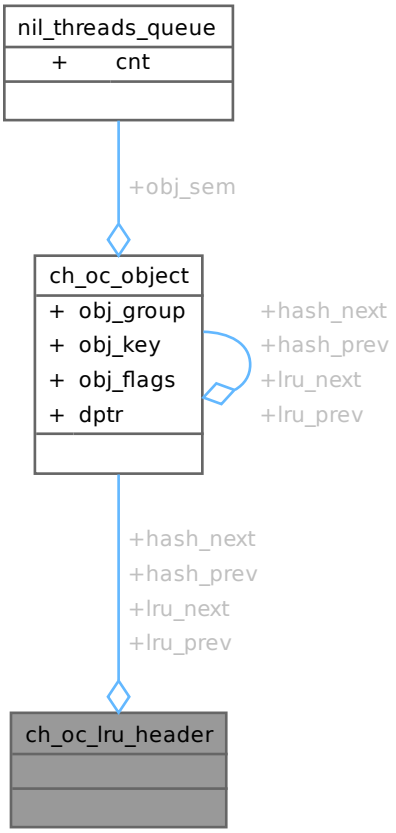
Previous in the collisions list.

7.15 ch_oc_lru_header Struct Reference

Structure representing an hash table element.

```
#include <chobjcaches.h>
```

Collaboration diagram for ch_oc_lru_header:



Data Fields

- oc_object_t * hash_next
  Next in the collisions list.
- oc_object_t * hash_prev
  Previous in the collisions list.
- oc_object_t * lru_next
  Next in the LRU list.
- oc_object_t * lru_prev
  Previous in the LRU list.

7.15.1 Detailed Description

Structure representing an hash table element.

7.15.2 Field Documentation

7.15.2.1 hash_next

```
oc_object_t* ch_oc_lru_header::hash_next
```

Next in the collisions list.

7.15.2.2 hash_prev

```
oc_object_t* ch_oc_lru_header::hash_prev
```

Previous in the collisions list.

7.15.2.3 lru_next

```
oc_object_t* ch_oc_lru_header::lru_next
```

Next in the LRU list.

7.15.2.4 lru_prev

```
oc_object_t* ch_oc_lru_header::lru_prev
```

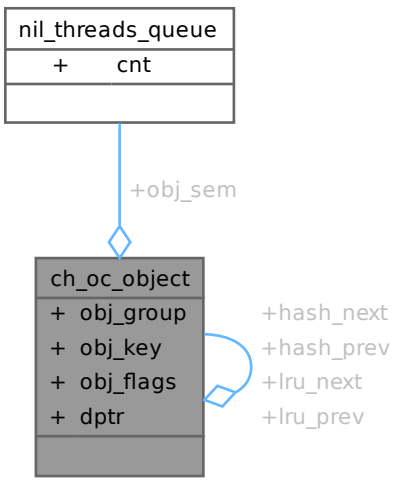
Previous in the LRU list.

7.16 ch_oc_object Struct Reference

Structure representing a cached object.

```
#include <chobjcaches.h>
```

Collaboration diagram for ch_oc_object:



Data Fields

- oc_object_t * hash_next
  Next in the collisions list.
- oc_object_t * hash_prev
  Previous in the collisions list.
- oc_object_t * lru_next
  Next in the LRU list.
- oc_object_t * lru_prev
  Previous in the LRU list.
- uint32_t obj_group
  Object group.
- uint32_t obj_key
  Object key.
- semaphore_t obj_sem
  Semaphore for object access.
- oc_flags_t obj_flags
  Object flags.
- void * dptr
  User pointer.

7.16.1 Detailed Description

Structure representing a cached object.

7.16.2 Field Documentation

7.16.2.1 hash_next

```
oc_object_t* ch_oc_object::hash_next
```

Next in the collisions list.

7.16.2.2 hash_prev

```
oc_object_t* ch_oc_object::hash_prev
```

Previous in the collisions list.

7.16.2.3 lru_next

```
oc_object_t* ch_oc_object::lru_next
```

Next in the LRU list.

7.16.2.4 lru_prev

```
oc_object_t* ch_oc_object::lru_prev
```

Previous in the LRU list.

7.16.2.5 obj_group

```
uint32_t ch_oc_object::obj_group
```

Object group.

7.16.2.6 obj_key

```
uint32_t ch_oc_object::obj_key
```

Object key.

7.16.2.7 obj_sem

```
semaphore_t ch_oc_object::obj_sem
```

Semaphore for object access.

7.16.2.8 obj_flags

```
oc_flags_t ch_oc_object::obj_flags
```

Object flags.

7.16.2.9 dptr

```
void* ch_oc_object::dptr
```

User pointer.

Note

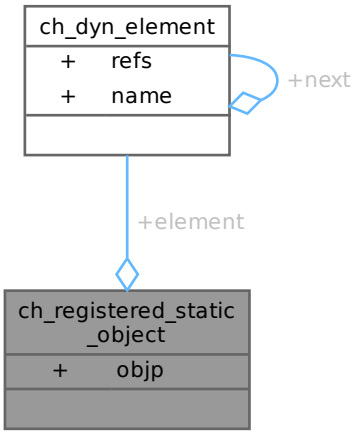
This pointer can be used to refer to external buffers, `chCacheObjectInit()` initializes it to `NULL`.

7.17 ch_registered_static_object Struct Reference

Type of a registered object.

`#include <chfactory.h>`

Collaboration diagram for ch_registered_static_object:



Data Fields

- dyn_element_t element
  List element of the registered object.
- void * objp
  Pointer to the object.

7.17.1 Detailed Description

Type of a registered object.

7.17.2 Field Documentation

7.17.2.1 element

`dyn_element_t ch_registered_static_object::element`

List element of the registered object.

7.17.2.2 objp

```
void* ch_registered_static_object::objp
```

Pointer to the object.

Note

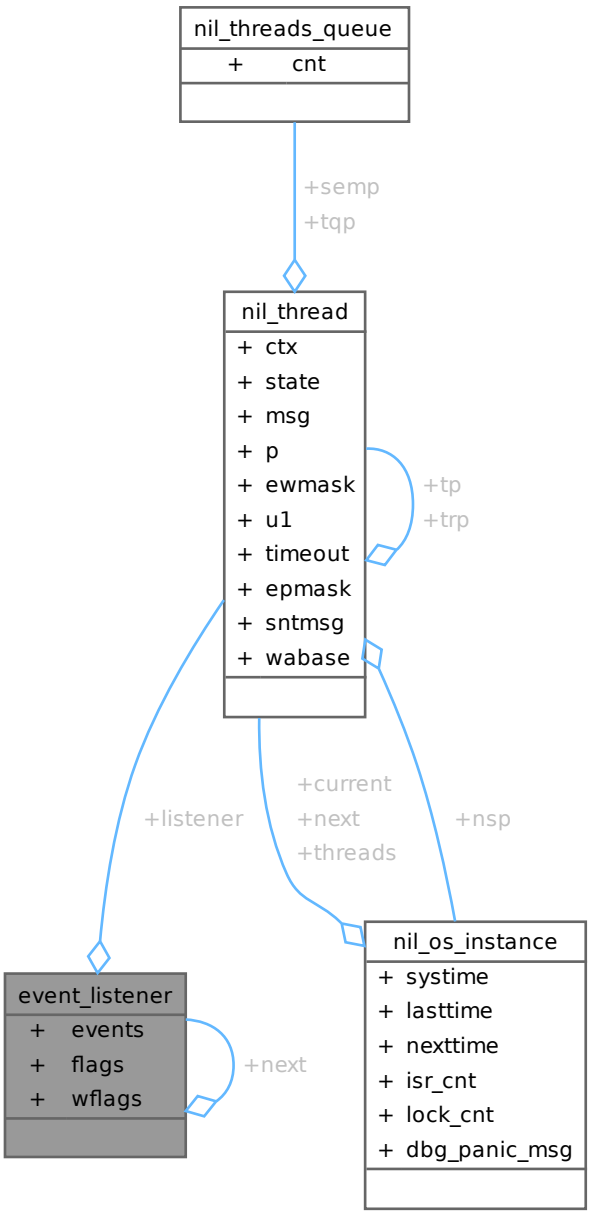
The type of the object is not stored in anyway.

7.18 event_listener Struct Reference

Event Listener structure.

```
#include <chevt.h>
```

Collaboration diagram for event_listener:



Data Fields

- event_listener_t * next

  Next Event Listener registered on the event source.
- thread_t * listener

  Thread interested in the event source.
- eventmask_t events

  Events to be set in the listening thread.

- eventflags_t flags

  Flags added to the listener by the event source.
- eventflags_t wflags

  Flags that this listener interested in.

7.18.1 Detailed Description

Event Listener structure.

7.18.2 Field Documentation

7.18.2.1 next

```
event_listener_t* event_listener::next
```

Next Event Listener registered on the event source.

7.18.2.2 listener

```
thread_t* event_listener::listener
```

Thread interested in the event source.

7.18.2.3 events

```
eventmask_t event_listener::events
```

Events to be set in the listening thread.

7.18.2.4 flags

```
eventflags_t event_listener::flags
```

Flags added to the listener by the event source.

7.18.2.5 wflags

```
eventflags_t event_listener::wflags
```

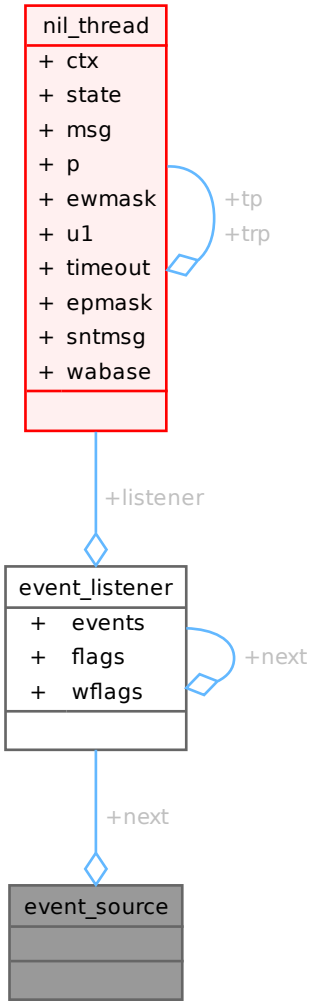
Flags that this listener interested in.

7.19 event_source Struct Reference

Event Source structure.

`#include <chevt.h>`

Collaboration diagram for event_source:



Data Fields

- event_listener_t * next

  First Event Listener registered on the Event Source.

7.19.1 Detailed Description

Event Source structure.

7.19.2 Field Documentation

7.19.2.1 next

```
event_listener_t* event_source::next
```

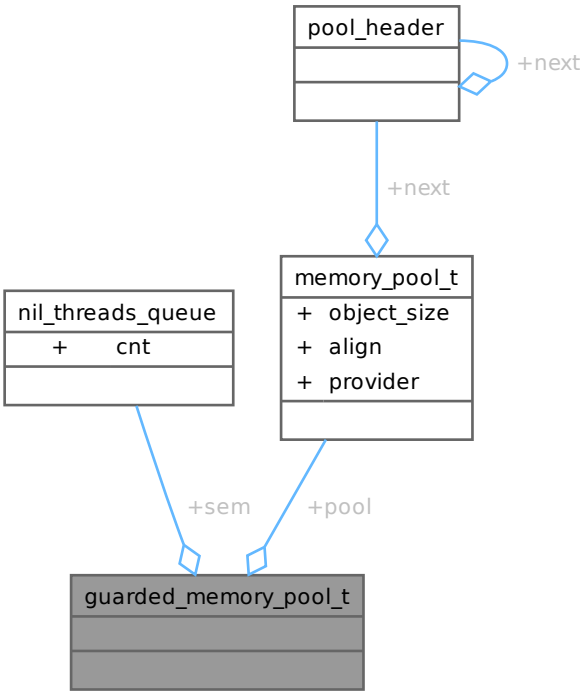
First Event Listener registered on the Event Source.

7.20 guarded_memory_pool_t Struct Reference

Guarded memory pool descriptor.

```
#include <chmempools.h>
```

Collaboration diagram for guarded_memory_pool_t:



Data Fields

- semaphore_t sem

  Counter semaphore guarding the memory pool.
- memory_pool_t pool

  The memory pool itself.

7.20.1 Detailed Description

Guarded memory pool descriptor.

7.20.2 Field Documentation

7.20.2.1 sem

```
semaphore_t guarded_memory_pool_t::sem
```

Counter semaphore guarding the memory pool.

7.20.2.2 pool

```
memory_pool_t guarded_memory_pool_t::pool
```

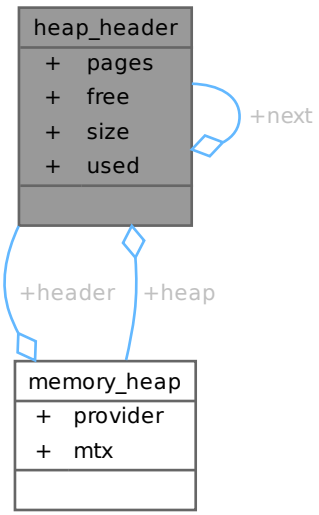
The memory pool itself.

7.21 heap_header Union Reference

Memory heap block header.

```
#include <chmemheaps.h>
```

Collaboration diagram for heap_header:



Data Fields

- struct {
  heap_header_t * next
  Next block in free list.
  size_t pages
  Size of the area in pages.
  } free

- struct {
  memory_heap_t * heap
  Block owner heap.
  size_t size
  Size of the area in bytes.
  } used

7.21.1 Detailed Description

Memory heap block header.

7.21.2 Field Documentation

7.21.2.1 next

```
heap_header_t* heap_header::next
```

Next block in free list.

7.21.2.2 pages

```
size_t heap_header::pages
```

Size of the area in pages.

7.21.2.3 [struct]

```
struct { ... } heap_header::free
```

7.21.2.4 heap

```
memory_heap_t* heap_header::heap
```

Block owner heap.

7.21.2.5 size

```
size_t heap_header::size
```

Size of the area in bytes.

7.21.2.6 [struct]

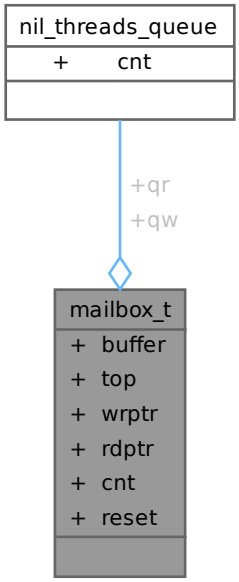
```
struct { ... } heap_header::used
```

7.22 mailbox_t Struct Reference

Structure representing a mailbox object.

```
#include <chmboxes.h>
```

Collaboration diagram for mailbox_t:



Data Fields

- msg_t * buffer
  Pointer to the mailbox buffer.
- msg_t * top
  Pointer to the location after the buffer.
- msg_t * wrptr
  Write pointer.
- msg_t * rdptr
  Read pointer.
- size_t cnt
  Messages in queue.
- bool reset
  True in reset state.
- threads_queue_t qw
  Queued writers.
- threads_queue_t qr
  Queued readers.

7.22.1 Detailed Description

Structure representing a mailbox object.

7.22.2 Field Documentation

7.22.2.1 buffer

```
msg_t* mailbox_t::buffer
```

Pointer to the mailbox buffer.

7.22.2.2 top

```
msg_t* mailbox_t::top
```

Pointer to the location after the buffer.

7.22.2.3 wrptr

```
msg_t* mailbox_t::wrptr
```

Write pointer.

7.22.2.4 rdptr

```
msg_t* mailbox_t::rdptr
```

Read pointer.

7.22.2.5 cnt

```
size_t mailbox_t::cnt
```

Messages in queue.

7.22.2.6 reset

```
bool mailbox_t::reset
```

True in reset state.

7.22.2.7 qw

```
threads_queue_t mailbox_t::qw
```

Queued writers.

7.22.2.8 qr

```
threads_queue_t mailbox_t::qr
```

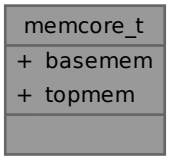
Queued readers.

7.23 memcore_t Struct Reference

Type of memory core object.

```
#include <chmemcore.h>
```

Collaboration diagram for memcore_t:



Data Fields

- uint8_t * basemem
  Next free address.
- uint8_t * topmem
  Final address.

7.23.1 Detailed Description

Type of memory core object.

7.23.2 Field Documentation

7.23.2.1 basemem

```
uint8_t* memcore_t::basemem
```

Next free address.

7.23.2.2 topmem

```
uint8_t* memcore_t::topmem
```

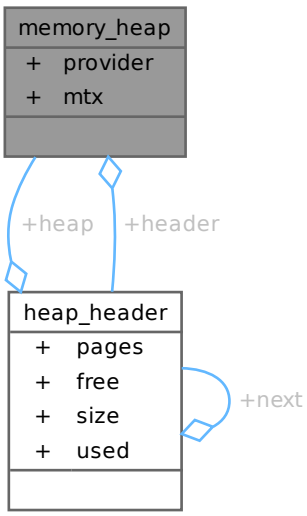
Final address.

7.24 memory_heap Struct Reference

Structure describing a memory heap.

`#include <chmemheaps.h>`

Collaboration diagram for memory_heap:



Data Fields

- memgetfunc2_t provider

  Memory blocks provider for this heap.
- heap_header_t header

  Free blocks list header.
- mutex_t mtx

  Heap access mutex.

7.24.1 Detailed Description

Structure describing a memory heap.

7.24.2 Field Documentation

7.24.2.1 provider

```
memgetfunc2_t memory_heap::provider
```

Memory blocks provider for this heap.

7.24.2.2 header

```
heap_header_t memory_heap::header
```

Free blocks list header.

7.24.2.3 mtx

```
mutex_t memory_heap::mtx
```

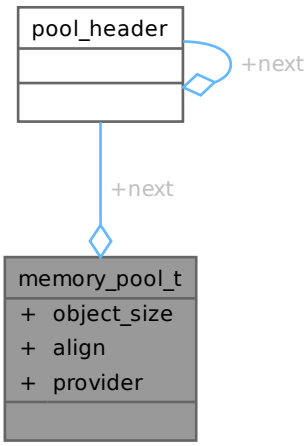
Heap access mutex.

7.25 memory_pool_t Struct Reference

Memory pool descriptor.

`#include <chmempools.h>`

Collaboration diagram for memory_pool_t:



Data Fields

- struct pool_header * next
  Pointer to the header.
- size_t object_size
  Memory pool objects size.
- unsigned align
  Required alignment.
- memgetfunc_t provider
  Memory blocks provider for this pool.

7.25.1 Detailed Description

Memory pool descriptor.

7.25.2 Field Documentation

7.25.2.1 next

`struct pool_header* memory_pool_t::next`

Pointer to the header.

7.25.2.2 object_size

```
size_t memory_pool_t::object_size
```

Memory pool objects size.

7.25.2.3 align

```
unsigned memory_pool_t::align
```

Required alignment.

7.25.2.4 provider

```
memgetfunc_t memory_pool_t::provider
```

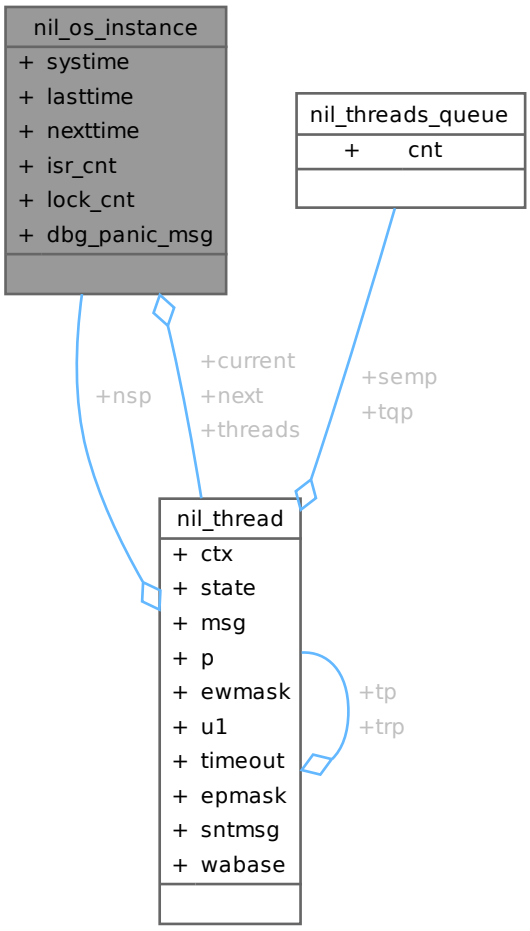
Memory blocks provider for this pool.

7.26 nil_os_instance Struct Reference

System data structure.

```
#include <ch.h>
```

Collaboration diagram for nil_os_instance:



Data Fields

- thread_t * current

  Pointer to the running thread.
- thread_t * next

  Pointer to the next thread to be executed.
- volatile systime_t systime

  System time.
- systime_t lasttime

  System time of the last tick event.
- systime_t nexttime

  Time of the next scheduled tick event.
- cnt_t isr_cnt

  ISR nesting level.
- cnt_t lock_cnt

  Lock nesting level.

- const char *volatile dbg_panic_msg

  Panic message.
- thread_t threads [CH_CFG_MAX_THREADS+1]

  Thread structures for all the defined threads.

7.26.1 Detailed Description

System data structure.

Note

This structure contain all the data areas used by the OS except stacks.

7.26.2 Field Documentation

7.26.2.1 current

```
thread_t* nil_os_instance::current
```

Pointer to the running thread.

7.26.2.2 next

```
thread_t* nil_os_instance::next
```

Pointer to the next thread to be executed.

Note

This pointer must point at the same thread pointed by `current` or to an higher priority thread if a switch is required.

7.26.2.3 systime

```
volatile systime_t nil_os_instance::systime
```

System time.

7.26.2.4 lasttime

```
systime_t nil_os_instance::lasttime
```

System time of the last tick event.

7.26.2.5 nexttime

```
systime_t nil_os_instance::nexttime
```

Time of the next scheduled tick event.

7.26.2.6 isr_cnt

```
cnt_t nil_os_instance::isr_cnt
```

ISR nesting level.

7.26.2.7 lock_cnt

```
cnt_t nil_os_instance::lock_cnt
```

Lock nesting level.

7.26.2.8 dbg_panic_msg

```
const char* volatile nil_os_instance::dbg_panic_msg
```

Panic message.

Note

This field is only present if some debug options have been activated.

Accesses to this pointer must never be optimized out so the field itself is declared volatile.

7.26.2.9 threads

```
thread_t nil_os_instance::threads[CH_CFG_MAX_THREADS+1]
```

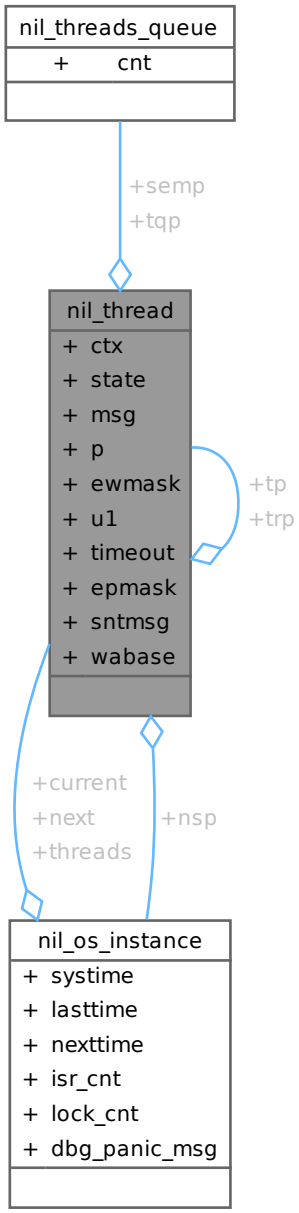
Thread structures for all the defined threads.

7.27 nil_thread Struct Reference

Structure representing a thread.

`#include <ch.h>`

Collaboration diagram for nil_thread:



Data Fields

- struct port_context ctx

Processor context.

- tstate_t state

  Thread state.

- union {
  - msg_t msg

    Wake-up/exit message.

  - void * p

    Generic pointer.

  - os_instance_t * nsp

    Pointer to nil base struct.

  - thread_reference_t * trp

    Pointer to thread reference.

  - threads_queue_t * tqp

    Pointer to thread queue.

  - thread_t * tp

    Pointer to thread.

  - semaphore_t * semp

    Pointer to semaphore.

  - eventmask_t ewmask

    Enabled events mask.

  } u1

- volatile sysinterval_t timeout

  Timeout counter, zero if disabled.

- eventmask_t epmask

  Pending events mask.

- msg_t sntmsg

  Sent message.

- stkalign_t * wabase

  Thread stack boundary.

7.27.1 Detailed Description

Structure representing a thread.

7.27.2 Field Documentation

7.27.2.1 ctx

```
struct port_context nil_thread::ctx
```

Processor context.

7.27.2.2 state

```
tstate_t nil_thread::state
```

Thread state.

7.27.2.3 msg

```
msg_t nil_thread::msg
```

Wake-up/exit message.

7.27.2.4 p

```
void* nil_thread::p
```

Generic pointer.

7.27.2.5 nsp

```
os_instance_t* nil_thread::nsp
```

Pointer to nil base struct.

7.27.2.6 trp

```
thread_reference_t* nil_thread::trp
```

Pointer to thread reference.

7.27.2.7 tqp

```
threads_queue_t* nil_thread::tqp
```

Pointer to thread queue.

7.27.2.8 tp

```
thread_t* nil_thread::tp
```

Pointer to thread.

7.27.2.9 semp

```
semaphore_t* nil_thread::semp
```

Pointer to semaphore.

7.27.2.10 ewmask

```
eventmask_t nil_thread::ewmask
```

Enabled events mask.

7.27.2.11 [union]

```
union { ... } nil_thread::u1
```

7.27.2.12 timeout

```
volatile sysinterval_t nil_thread::timeout
```

Timeout counter, zero if disabled.

7.27.2.13 epmask

```
eventmask_t nil_thread::epmask
```

Pending events mask.

7.27.2.14 sntmsg

```
msg_t nil_thread::sntmsg
```

Sent message.

7.27.2.15 wabase

```
stkalign_t* nil_thread::wabase
```

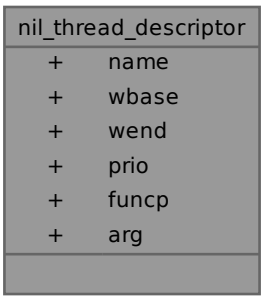
Thread stack boundary.

7.28 nil_thread_descriptor Struct Reference

Structure representing a thread descriptor.

```
#include <ch.h>
```

Collaboration diagram for nil_thread_descriptor:



Data Fields

- const char * name
  Thread name, for debugging.
- stkalign_t * wbase
  Thread working area base.
- stkalign_t * wend
  Thread working area end.
- tprio_t prio
  Thread priority slot.
- tfunc_t funcp
  Thread function.
- void * arg
  Thread function argument.

7.28.1 Detailed Description

Structure representing a thread descriptor.

7.28.2 Field Documentation

7.28.2.1 name

```
const char* nil_thread_descriptor::name
```

Thread name, for debugging.

7.28.2.2 wbase

```
stkalign_t* nil_thread_descriptor::wbase
```

Thread working area base.

7.28.2.3 wend

```
stkalign_t* nil_thread_descriptor::wend
```

Thread working area end.

7.28.2.4 prio

```
tprio_t nil_thread_descriptor::prio
```

Thread priority slot.

7.28.2.5 funcp

```
tfunc_t nil_thread_descriptor::funcp
```

Thread function.

7.28.2.6 arg

```
void* nil_thread_descriptor::arg
```

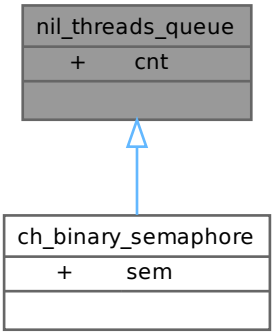
Thread function argument.

7.29 nil_threads_queue Struct Reference

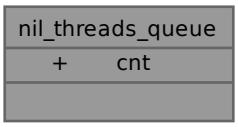
Structure representing a queue of threads.

```
#include <ch.h>
```

Inheritance diagram for nil_threads_queue:



Collaboration diagram for nil_threads_queue:



Data Fields

- volatile cnt_t cnt

  Threads Queue counter.

7.29.1 Detailed Description

Structure representing a queue of threads.

7.29.2 Field Documentation

7.29.2.1 cnt

```
volatile cnt_t nil_threads_queue::cnt
```

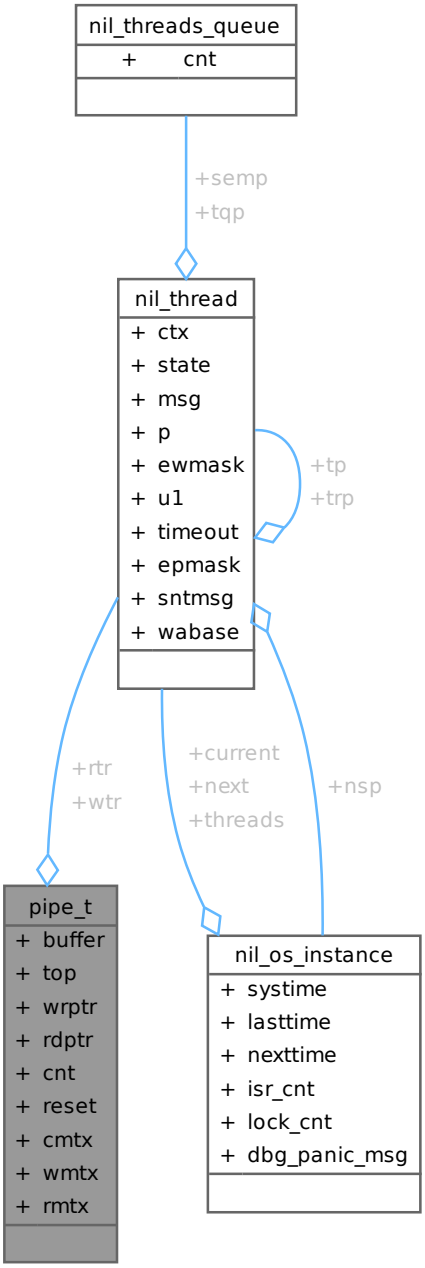
Threads Queue counter.

7.30 pipe_t Struct Reference

Structure representing a pipe object.

```
#include <chpipes.h>
```

Collaboration diagram for pipe_t:



Data Fields

- uint8_t * buffer

  Pointer to the pipe buffer.
- uint8_t * top

  Pointer to the location after the buffer.
- uint8_t * wrptr

Write pointer.

- uint8_t * rdptr

  Read pointer.
- size_t cnt

  Bytes in the pipe.
- bool reset

  True if in reset state.
- thread_reference_t wtr

  Waiting writer.
- thread_reference_t rtr

  Waiting reader.
- mutex_t cmtx

  Common access mutex.
- mutex_t wmtx

  Write access mutex.
- mutex_t rmtx

  Read access mutex.

7.30.1 Detailed Description

Structure representing a pipe object.

7.30.2 Field Documentation

7.30.2.1 buffer

```
uint8_t* pipe_t::buffer
```

Pointer to the pipe buffer.

7.30.2.2 top

```
uint8_t* pipe_t::top
```

Pointer to the location after the buffer.

7.30.2.3 wrptr

```
uint8_t* pipe_t::wrptr
```

Write pointer.

7.30.2.4 rdptr

```
uint8_t* pipe_t::rdptr
```

Read pointer.

7.30.2.5 cnt

```
size_t pipe_t::cnt
```

Bytes in the pipe.

7.30.2.6 reset

```
bool pipe_t::reset
```

True if in reset state.

7.30.2.7 wtr

```
thread_reference_t pipe_t::wtr
```

Waiting writer.

7.30.2.8 rtr

```
thread_reference_t pipe_t::rtr
```

Waiting reader.

7.30.2.9 cmtx

```
mutex_t pipe_t::cmtx
```

Common access mutex.

7.30.2.10 wmtx

```
mutex_t pipe_t::wmtx
```

Write access mutex.

7.30.2.11 rmtx

```
mutex_t pipe_t::rmtx
```

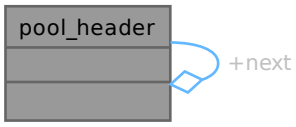
Read access mutex.

7.31 pool_header Struct Reference

Memory pool free object header.

`#include <chmempools.h>`

Collaboration diagram for pool_header:



Data Fields

- struct pool_header * next
  Pointer to the next pool header in the list.

7.31.1 Detailed Description

Memory pool free object header.

7.31.2 Field Documentation

7.31.2.1 next

`struct pool_header* pool_header::next`

Pointer to the next pool header in the list.

Chapter 8

File Documentation

8.1 main.dox File Reference

8.2 ccportab.h File Reference

Compiler portability layer.

Macros

Compiler abstraction macros

- #define CC_SECTION(s)
  Allocates a variable or function to a specific section.
- #define CC_WEAK __attribute__((weak))
  Marks a function or variable as a weak symbol.
- #define CC_USED __attribute__((used))
  Marks a function or variable as used.
- #define CC_ALIGN_DATA(n)
  Enforces alignment of the variable declared afterward.
- #define CC_ALIGN_CODE(n)
  Enforces alignment of a function declared afterward.
- #define CC_PACK __attribute__((packed))
  Enforces packing of the structure declared afterward.
- #define CC_NO_INLINE __attribute__((noinline))
  Marks a function as not inlineable.
- #define CC_FORCE_INLINE __attribute__((always_inline))
  Enforces a function inline.
- #define CC_NO_RETURN __attribute__((noreturn))
  Marks a function as non-returning.
- #define CC_ROMCONST const
  Enforces a variable in a ROM area.
- #define CC_LIKELY(x)
  Marks a boolean expression as likely true.
- #define CC_UNLIKELY(x)
  Marks a boolean expression as likely false.

8.2.1 Detailed Description

Compiler portability layer.

8.3 nil.dox File Reference

8.4 ch.h File Reference

Nil RTOS main header file.

```
#include "chtypes.h"
#include "chconf.h"
#include "chlicense.h"
#include "chcore.h"
#include "chsem.h"
#include "chevt.h"
#include "chmsg.h"
#include "chlib.h"
```

Data Structures

- struct nil_threads_queue

  Structure representing a queue of threads.
- struct nil_thread_descriptor

  Structure representing a thread descriptor.
- struct nil_thread

  Structure representing a thread.
- struct nil_os_instance

  System data structure.

Macros

- #define __CHIBIOS_NIL__

  ChibiOS/NIL identification macro.
- #define CH_KERNEL_STABLE 1

  Stable release flag.
- #define CH_CFG_ST_TIMEDELTA 0
- #define CH_CFG_USE_MESSAGES FALSE
- #define NIL_DBG_ENABLED TRUE
- #define THD_IDLE_BASE (&__main_thread_stack_base__)
- #define THD_IDLE_END (&__main_thread_stack_end__)
- #define CH_PORT_SUPPORTS_RECURSIVE_LOCKS FALSE
- #define __dbg_enter_lock()
- #define __dbg_leave_lock()
- #define __CH_STRINGIFY(a)

  Utility to make the parameter a quoted string.
- #define THD_WORKING_AREA_END(wa)

  Returns the top address of a working area.

- #define __dbg_enter_lock()
- #define __dbg_leave_lock()
- #define __dbg_check_disable()
- #define __dbg_check_suspend()
- #define __dbg_check_enable()
- #define __dbg_check_lock()
- #define __dbg_check_unlock()
- #define __dbg_check_lock_from_isr()
- #define __dbg_check_unlock_from_isr()
- #define __dbg_check_enter_isr()
- #define __dbg_check_leave_isr()
- #define chDbgCheckClassI()
- #define chDbgCheckClassS()

ChibiOS/NIL version identification

- #define CH_KERNEL_VERSION "4.1.3"
  Kernel version string.
- #define CH_KERNEL_MAJOR 4
  Kernel version major number.
- #define CH_KERNEL_MINOR 1
  Kernel version minor number.
- #define CH_KERNEL_PATCH 3
  Kernel version patch number.

Constants for configuration options

- #define FALSE 0
  Generic 'false' preprocessor boolean constant.
- #define TRUE 1
  Generic 'true' preprocessor boolean constant.

Wakeup messages

- #define MSG_OK (msg_t)0
  OK wakeup message.
- #define MSG_TIMEOUT (msg_t)-1
  Wake-up caused by a timeout condition.
- #define MSG_RESET (msg_t)-2
  Wake-up caused by a reset condition.

Special time constants

- #define TIME_IMMEDIATE ((sysinterval_t)-1)
  Zero time specification for some functions with a timeout specification.
- #define TIME_INFINITE ((sysinterval_t)0)
  Infinite time specification for all functions with a timeout specification.
- #define TIME_MAX_INTERVAL ((sysinterval_t)-2)
  Maximum interval constant usable as timeout.
- #define TIME_MAX_SYSTIME ((systime_t)-1)
  Maximum system of system time before it wraps.

Thread state related macros

- #define NIL_STATE_WTSTART (tstate_t)0

Thread not yet started or terminated.
- #define NIL_STATE_READY (tstate_t)1
  Thread ready or executing.
- #define NIL_STATE_SLEEPING (tstate_t)2
  Thread sleeping.
- #define NIL_STATE_SUSPENDED (tstate_t)3
  Thread suspended.
- #define NIL_STATE_WTEXIT (tstate_t)4
  Waiting a thread.
- #define NIL_STATE_WTQUEUE (tstate_t)5
  On queue or semaph.
- #define NIL_STATE_WTOREVT (tstate_t)6
  Waiting for events.
- #define NIL_STATE_WTANDEVT (tstate_t)7
  Waiting for events.
- #define NIL_STATE_SNDMSGQ (tstate_t)8
  Sending a message, in queue.
- #define NIL_STATE_WTMSG (tstate_t)1
  Waiting for a message.
- #define NIL_STATE_FINAL (tstate_t)1
  Thread terminated.
- #define NIL_THD_IS_WTSTART(tp)
- #define NIL_THD_IS_READY(tp)
- #define NIL_THD_IS_SLEEPING(tp)
- #define NIL_THD_IS_SUSPENDED(tp)
- #define NIL_THD_IS_WTEXIT(tp)
- #define NIL_THD_IS_WTQUEUE(tp)
- #define NIL_THD_IS_WTOREVT(tp)
- #define NIL_THD_IS_WTANDEVT(tp)
- #define NIL_THD_IS_SNDMSGQ(tp)
- #define NIL_THD_IS_WTMSG(tp)
- #define NIL_THD_IS_FINAL(tp)
- #define CH_STATE_NAMES

RT options not existing in NIL

- #define CH_CFG_USE_REGISTRY FALSE

Threads tables definition macros

- #define THD_TABLE_BEGIN const thread_descriptor_t nil_thd_configs[ ] = {
  Start of user threads table.
- #define THD_TABLE_THREAD(_prio, _name, _wap, _funcp, _arg)
  Entry of user threads table.
- #define THD_TABLE_END
  End of user threads table.

Memory alignment support macros

- #define MEM_ALIGN_MASK(a)
  Alignment mask constant.
- #define MEM_ALIGN_PREV(p, a)
  Aligns to the previous aligned memory address.
- #define MEM_ALIGN_NEXT(p, a)
  Aligns to the new aligned memory address.
- #define MEM_IS_ALIGNED(p, a)
  Returns whatever a pointer or memory size is aligned.
- #define MEM_IS_VALID_ALIGNMENT(a)

Returns whatever a constant is a valid alignment.

Working Areas

- #define THD_WORKING_AREA_SIZE(n)
  Calculates the total Working Area size.
- #define THD_WORKING_AREA(s, n)
  Static working area allocation.

Threads abstraction macros

- #define THD_FUNCTION(tname, arg)
  Thread declaration macro.

ISRs abstraction macros

- #define CH_IRQ_IS_VALID_PRIORITY(prio)
  Priority level validation macro.
- #define CH_IRQ_IS_VALID_KERNEL_PRIORITY(prio)
  Priority level validation macro.
- #define CH_IRQ_PROLOGUE()
  IRQ handler enter code.
- #define CH_IRQ_EPILOGUE()
  IRQ handler exit code.
- #define CH_IRQ_HANDLER(id)
  Standard normal IRQ handler declaration.

Fast ISRs abstraction macros

- #define CH_FAST_IRQ_HANDLER(id)
  Standard fast IRQ handler declaration.

Time conversion utilities

- #define TIME_S2I(secs)
  Seconds to time interval.
- #define TIME_MS2I(msecs)
  Milliseconds to time interval.
- #define TIME_US2I(usecs)
  Microseconds to time interval.
- #define TIME_I2S(interval)
  Time interval to seconds.
- #define TIME_I2MS(interval)
  Time interval to milliseconds.
- #define TIME_I2US(interval)
  Time interval to microseconds.

Threads queues

- #define __THREADS_QUEUE_DATA(name)
  Data part of a static threads queue object initializer.
- #define THREADS_QUEUE_DECL(name)
  Static threads queue object initializer.

Macro Functions

- #define chSysGetRealtimeCounterX()
  Returns the current value of the system real time counter.
- #define chSysDisable()
  Raises the system interrupt priority mask to the maximum level.
- #define chSysSuspend()
  Raises the system interrupt priority mask to system level.
- #define chSysEnable()
  Lowers the system interrupt priority mask to user level.
- #define chSysLock()
  Enters the kernel lock state.
- #define chSysUnlock()
  Leaves the kernel lock state.
- #define chSysLockFromISR()
  Enters the kernel lock state from within an interrupt handler.
- #define chSysUnlockFromISR()
  Leaves the kernel lock state from within an interrupt handler.
- #define chSchGoSleepS(newstate)
  Puts the current thread to sleep into the specified state.
- #define chSchWakeupS(ntp, msg)
  Wakes up a thread.
- #define chSchIsRescRequiredI()
  Evaluates if a reschedule is required.
- #define chThdGetSelfX()
  Returns a pointer to the current `thread_t`.
- #define chThdGetPriorityX(void)
  Returns the current thread priority.
- #define chThdResumeS(trp, msg)
  Wakes up a thread waiting on a thread reference object.
- #define chThdSleepSeconds(secs)
  Delays the invoking thread for the specified number of seconds.
- #define chThdSleepMilliseconds(msecs)
  Delays the invoking thread for the specified number of milliseconds.
- #define chThdSleepMicroseconds(usecs)
  Delays the invoking thread for the specified number of microseconds.
- #define chThdSleepS(timeout)
  Suspends the invoking thread for the specified time.
- #define chThdSleepUntilS(abstime)
  Suspends the invoking thread until the system time arrives to the specified value.
- #define chThdQueueObjectInit(tqp)
  Initializes a threads queue object.
- #define chThdQueueIsEmptyI(tqp)
  Evaluates to `true` if the specified queue is empty.
- #define chVTGetSystemTimeX()
  Current system time.
- #define chVTTimeElapsedSinceX(start)
  Returns the elapsed time since the specified start time.
- #define chVTIsSystemTimeWithinX(start, end)
  Checks if the current system time is within the specified time window.
- #define chTimeAddX(systime, interval)
  Adds an interval to a system time returning a system time.
- #define chTimeDiffX(start, end)
  Subtracts two system times returning an interval.
- #define chDbgCheck(c)
  Function parameters check.
- #define chDbgAssert(c, r)
  Condition assertion.

Typedefs

- typedef uint32_t systime_t

  Type of system time.
- typedef uint32_t sysinterval_t

  Type of time interval.
- typedef uint64_t time_conv_t

  Type of time conversion variable.
- typedef struct nil_os_instance os_instance_t

  Type of a structure representing the system.
- typedef void(∗ tfunc_t) (void ∗p)

  Thread function.
- typedef struct nil_thread_descriptor thread_descriptor_t

  Type of a thread descriptor.
- typedef struct nil_thread thread_t

  Type of a structure representing a thread.
- typedef thread_t ∗ thread_reference_t

  Type of a thread reference.
- typedef struct nil_threads_queue threads_queue_t

  Type of a queue of threads.
- typedef threads_queue_t semaphore_t

  Type of a structure representing a semaphore.

Kernel types

- typedef port_rtcnt_t rtcnt_t
- typedef port_syssts_t syssts_t
- typedef port_stkalign_t stkalign_t
- typedef uint8_t tstate_t
- typedef uint32_t tprio_t
- typedef int32_t msg_t
- typedef int32_t eventid_t
- typedef uint32_t eventmask_t
- typedef uint32_t eventflags_t
- typedef int32_t cnt_t
- typedef uint32_t ucnt_t

Functions

- thread_t ∗ nil_find_thread (tstate_t state, void ∗p)

  Retrieves the highest priority thread in the specified state and associated to the specified object.
- cnt_t nil_ready_all (void ∗p, cnt_t cnt, msg_t msg)

  Puts in ready state all thread matching the specified status and associated object.
- void chSysInit (void)

  Initializes the kernel.
- void chSysHalt (const char ∗reason)

  Halts the system.
- void chSysTimerHandlerI (void)

  Time management handler.
- void chSysUnconditionalLock (void)

  Unconditionally enters the kernel lock state.
- void chSysUnconditionalUnlock (void)

  Unconditionally leaves the kernel lock state.

- syssts_t chSysGetStatusAndLockX (void)
  Returns the execution status and enters a critical zone.
- bool chSysIsCounterWithinX (rtcnt_t cnt, rtcnt_t start, rtcnt_t end)
  Realtime window test.
- void chSysPolledDelayX (rtcnt_t cycles)
  Polled delay.
- void chSysRestoreStatusX (syssts_t sts)
  Restores the specified execution status and leaves a critical zone.
- thread_t * chSchReadyI (thread_t *tp, msg_t msg)
  Makes the specified thread ready for execution.
- bool chSchIsPreemptionRequired (void)
  Evaluates if preemption is required.
- void chSchDoPreemption (void)
  Switches to the first thread on the runnable queue.
- void chSchRescheduleS (void)
  Reschedules if needed.
- msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)
  Puts the current thread to sleep into the specified state with timeout specification.
- bool chTimeIsInRangeX (systime_t time, systime_t start, systime_t end)
  Checks if the specified time is within the specified time range.
- thread_t * chThdCreateI (const thread_descriptor_t *tdp)
  Creates a new thread into a static memory area.
- thread_t * chThdCreate (const thread_descriptor_t *tdp)
  Creates a new thread into a static memory area.
- void chThdExit (msg_t msg)
  Terminates the current thread.
- msg_t chThdWait (thread_t *tp)
  Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
- msg_t chThdSuspendTimeoutS (thread_reference_t *trp, sysinterval_t timeout)
  Sends the current thread sleeping and sets a reference variable.
- void chThdResumeI (thread_reference_t *trp, msg_t msg)
  Wakes up a thread waiting on a thread reference object.
- void chThdResume (thread_reference_t *trp, msg_t msg)
  Wakes up a thread waiting on a thread reference object.
- void chThdSleep (sysinterval_t timeout)
  Suspends the invoking thread for the specified time.
- void chThdSleepUntil (systime_t abstime)
  Suspends the invoking thread until the system time arrives to the specified value.
- msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)
  Enqueues the caller thread on a threads queue object.
- void chThdDoDequeueNextI (threads_queue_t *tqp, msg_t msg)
  Dequeues and wakes up one thread from the threads queue object.
- void chThdDequeueNextI (threads_queue_t *tqp, msg_t msg)
  Dequeues and wakes up one thread from the threads queue object, if any.
- void chThdDequeueAllI (threads_queue_t *tqp, msg_t msg)
  Dequeues and wakes up all threads from the threads queue object.

8.4.1 Detailed Description

Nil RTOS main header file.

This header includes all the required kernel headers so it is the only header you usually need to include in your application.

8.5 chevt.h File Reference

Nil RTOS events header file.

Data Structures

- struct event_listener

  Event Listener structure.
- struct event_source

  Event Source structure.

Macros

- #define ALL_EVENTS ((eventmask_t)-1)

  All events allowed mask.
- #define EVENT_MASK(eid)

  Returns an event mask from an event identifier.
- #define _EVENTSOURCE_DATA(name)

  Data part of a static event source initializer.
- #define EVENTSOURCE_DECL(name)

  Static event source initializer.

Macro Functions

- #define chEvtObjectInit(esp)

  Initializes an Event Source.
- #define chEvtRegisterMask(esp, elp, events)

  Registers an Event Listener on an Event Source.
- #define chEvtRegister(esp, elp, event)

  Registers an Event Listener on an Event Source.
- #define chEvtIsListeningI(esp)

  Verifies if there is at least one `event_listener_t` registered.
- #define chEvtBroadcast(esp)

  Signals all the Event Listeners registered on the specified Event Source.
- #define chEvtBroadcastI(esp)

  Signals all the Event Listeners registered on the specified Event Source.
- #define chEvtAddEventsI(events)

  *Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.*
- #define chEvtGetEventsX(void)

  Returns the events mask.
- #define chEvtWaitOne(events)

  Waits for exactly one of the specified events.
- #define chEvtWaitAny(events)

  Waits for any of the specified events.
- #define chEvtWaitAll(events)

  Waits for all the specified events.

Typedefs

- typedef struct event_listener event_listener_t
- typedef struct event_source event_source_t

  Event Source structure.
- typedef void(* evhandler_t) (eventid_t id)

  Event Handler callback function.

Functions

- void chEvtRegisterMaskWithFlags (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)

  Registers an Event Listener on an Event Source.
- void chEvtUnregister (event_source_t *esp, event_listener_t *elp)

  Unregisters an Event Listener from its Event Source.
- eventmask_t chEvtGetAndClearEventsI (eventmask_t events)

  Clears the pending events specified in the events mask.
- eventmask_t chEvtGetAndClearEvents (eventmask_t events)

  Clears the pending events specified in the events mask.
- eventmask_t chEvtAddEvents (eventmask_t events)

  *Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.*
- eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)

  Returns the flags associated to an `event_listener_t`.
- eventflags_t chEvtGetAndClearFlagsI (event_listener_t *elp)

  Returns the unmasked flags associated to an `event_listener_t`.
- void chEvtSignal (thread_t *tp, eventmask_t events)

  Adds a set of event flags directly to the specified `thread_t`.
- void chEvtSignalI (thread_t *tp, eventmask_t events)

  Adds a set of event flags directly to the specified `thread_t`.
- void chEvtBroadcastFlags (event_source_t *esp, eventflags_t flags)

  Signals all the Event Listeners registered on the specified Event Source.
- void chEvtBroadcastFlagsI (event_source_t *esp, eventflags_t flags)

  Signals all the Event Listeners registered on the specified Event Source.
- void chEvtDispatch (const evhandler_t *handlers, eventmask_t events)

  Invokes the event handlers associated to an event flags mask.
- eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)

  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, sysinterval_t timeout)

  Waits for any of the specified events.
- eventmask_t chEvtWaitAllTimeout (eventmask_t mask, sysinterval_t timeout)

  Waits for all the specified events.

8.5.1 Detailed Description

Nil RTOS events header file.

8.6 chmsg.h File Reference

Nil RTOS synchronous messages header file.

Macros

Macro Functions

- #define chMsgWaitS()

  Suspends the thread and waits for an incoming message.
- #define chMsgGet(tp)

  Returns the message carried by the specified thread.
- #define chMsgReleaseS(tp, msg)

  Releases the thread waiting on top of the messages queue.

Functions

- msg_t chMsgSend (thread_t *tp, msg_t msg)
  Sends a message to the specified thread.
- thread_t * chMsgWait (void)
  Suspends the thread and waits for an incoming message.
- thread_t * chMsgWaitTimeout (sysinterval_t timeout)
  Suspends the thread and waits for an incoming message or a timeout to occur.
- thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)
  Suspends the thread and waits for an incoming message or a timeout to occur.
- void chMsgRelease (thread_t *tp, msg_t msg)
  Releases a sender thread specifying a response message.

8.6.1 Detailed Description

Nil RTOS synchronous messages header file.

8.7 chsem.h File Reference

Nil RTOS semaphores header file.

Macros

Semaphores macros

- #define __SEMAPHORE_DATA(name, n)
  Data part of a static semaphore initializer.
- #define SEMAPHORE_DECL(name, n)
  Static semaphore initializer.

Macro Functions

- #define chSemObjectInit(sp, n)
  Initializes a semaphore with the specified counter value.
- #define chSemReset(sp, n)
  Performs a reset operation on the semaphore.
- #define chSemResetI(sp, n)
  Performs a reset operation on the semaphore.
- #define chSemWait(sp)
  Performs a wait operation on a semaphore.
- #define chSemWaitS(sp)
  Performs a wait operation on a semaphore.
- #define chSemFastWaitI(sp)
  Decreases the semaphore counter.
- #define chSemFastSignalI(sp)
  Increases the semaphore counter.
- #define chSemGetCounterI(sp)
  Returns the semaphore counter current value.

Functions

- msg_t chSemWaitTimeout (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- msg_t chSemWaitTimeoutS (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- void chSemSignal (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemSignalI (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemResetWithMessage (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
- void chSemResetWithMessageI (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.

8.7.1 Detailed Description

Nil RTOS semaphores header file.

8.8 ch.c File Reference

Nil RTOS main source file.

```
#include "ch.h"
```

Functions

- thread_t * nil_find_thread (tstate_t state, void *p)
  Retrieves the highest priority thread in the specified state and associated to the specified object.
- cnt_t nil_ready_all (void *p, cnt_t cnt, msg_t msg)
  Puts in ready state all thread matching the specified status and associated object.
- void __dbg_check_disable (void)
  Guard code for `chSysDisable()`.
- void __dbg_check_suspend (void)
  Guard code for `chSysSuspend()`.
- void __dbg_check_enable (void)
  Guard code for `chSysEnable()`.
- void __dbg_check_lock (void)
  Guard code for `chSysLock()`.
- void __dbg_check_unlock (void)
  Guard code for `chSysUnlock()`.
- void __dbg_check_lock_from_isr (void)
  Guard code for `chSysLockFromIsr()`.
- void __dbg_check_unlock_from_isr (void)
  Guard code for `chSysUnlockFromIsr()`.
- void __dbg_check_enter_isr (void)
  Guard code for `CH_IRQ_PROLOGUE()`.

- void __dbg_check_leave_isr (void)

  Guard code for `CH_IRQ_EPILOGUE()`.
- void chDbgCheckClassI (void)

  I-class functions context check.
- void chDbgCheckClassS (void)

  S-class functions context check.
- void chSysInit (void)

  Initializes the kernel.
- void chSysHalt (const char *reason)

  Halts the system.
- void chSysTimerHandlerI (void)

  Time management handler.
- void chSysUnconditionalLock (void)

  Unconditionally enters the kernel lock state.
- void chSysUnconditionalUnlock (void)

  Unconditionally leaves the kernel lock state.
- syssts_t chSysGetStatusAndLockX (void)

  Returns the execution status and enters a critical zone.
- void chSysRestoreStatusX (syssts_t sts)

  Restores the specified execution status and leaves a critical zone.
- bool chSysIsCounterWithinX (rtcnt_t cnt, rtcnt_t start, rtcnt_t end)

  Realtime window test.
- void chSysPolledDelayX (rtcnt_t cycles)

  Polled delay.
- thread_t * chSchReadyI (thread_t *tp, msg_t msg)

  Makes the specified thread ready for execution.
- bool chSchIsPreemptionRequired (void)

  Evaluates if preemption is required.
- void chSchDoPreemption (void)

  Switches to the first thread on the runnable queue.
- void chSchRescheduleS (void)

  Reschedules if needed.
- msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)

  Puts the current thread to sleep into the specified state with timeout specification.
- bool chTimeIsInRangeX (systime_t time, systime_t start, systime_t end)

  Checks if the specified time is within the specified time range.
- thread_t * chThdCreateI (const thread_descriptor_t *tdp)

  Creates a new thread into a static memory area.
- thread_t * chThdCreate (const thread_descriptor_t *tdp)

  Creates a new thread into a static memory area.
- void chThdExit (msg_t msg)

  Terminates the current thread.
- msg_t chThdWait (thread_t *tp)

  Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
- msg_t chThdSuspendTimeoutS (thread_reference_t *trp, sysinterval_t timeout)

  Sends the current thread sleeping and sets a reference variable.
- void chThdResumeI (thread_reference_t *trp, msg_t msg)

  Wakes up a thread waiting on a thread reference object.
- void chThdResume (thread_reference_t *trp, msg_t msg)

  Wakes up a thread waiting on a thread reference object.
- void chThdSleep (sysinterval_t timeout)

Suspends the invoking thread for the specified time.

- void chThdSleepUntil (systime_t abstime)

  Suspends the invoking thread until the system time arrives to the specified value.

- msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)

  Enqueues the caller thread on a threads queue object.

- void chThdDoDequeueNextI (threads_queue_t *tqp, msg_t msg)

  Dequeues and wakes up one thread from the threads queue object.

- void chThdDequeueNextI (threads_queue_t *tqp, msg_t msg)

  Dequeues and wakes up one thread from the threads queue object, if any.

- void chThdDequeueAllI (threads_queue_t *tqp, msg_t msg)

  Dequeues and wakes up all threads from the threads queue object.

Variables

- os_instance_t nil

  System data structures.

8.8.1 Detailed Description

Nil RTOS main source file.

8.9 chevt.c File Reference

Nil RTOS events source file.

```
#include "ch.h"
```

Functions

- void chEvtRegisterMaskWithFlags (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)

  Registers an Event Listener on an Event Source.

- void chEvtUnregister (event_source_t *esp, event_listener_t *elp)

  Unregisters an Event Listener from its Event Source.

- eventmask_t chEvtGetAndClearEventsI (eventmask_t events)

  Clears the pending events specified in the events mask.

- eventmask_t chEvtGetAndClearEvents (eventmask_t events)

  Clears the pending events specified in the events mask.

- eventmask_t chEvtAddEvents (eventmask_t events)

  *Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.*

- void chEvtBroadcastFlagsI (event_source_t *esp, eventflags_t flags)

  Signals all the Event Listeners registered on the specified Event Source.

- eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)

  Returns the flags associated to an `event_listener_t`.

- void chEvtSignal (thread_t *tp, eventmask_t events)

Adds a set of event flags directly to the specified `thread_t`.

- void chEvtSignalI (thread_t *tp, eventmask_t events)

  Adds a set of event flags directly to the specified `thread_t`.
- void chEvtBroadcastFlags (event_source_t *esp, eventflags_t flags)

  Signals all the Event Listeners registered on the specified Event Source.
- eventflags_t chEvtGetAndClearFlagsI (event_listener_t *elp)

  Returns the unmasked flags associated to an `event_listener_t`.
- void chEvtDispatch (const evhandler_t *handlers, eventmask_t events)

  Invokes the event handlers associated to an event flags mask.
- eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)

  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, sysinterval_t timeout)

  Waits for any of the specified events.
- eventmask_t chEvtWaitAllTimeout (eventmask_t mask, sysinterval_t timeout)

  Waits for all the specified events.

8.9.1 Detailed Description

Nil RTOS events source file.

8.10 chmsg.c File Reference

Nil RTOS synchronous messages source file.

```
#include "ch.h"
```

Functions

- msg_t chMsgSend (thread_t *tp, msg_t msg)

  Sends a message to the specified thread.
- thread_t * chMsgWait (void)

  Suspends the thread and waits for an incoming message.
- thread_t * chMsgWaitTimeout (sysinterval_t timeout)

  Suspends the thread and waits for an incoming message or a timeout to occur.
- thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)

  Suspends the thread and waits for an incoming message or a timeout to occur.
- void chMsgRelease (thread_t *tp, msg_t msg)

  Releases a sender thread specifying a response message.

8.10.1 Detailed Description

Nil RTOS synchronous messages source file.

8.11 chsem.c File Reference

Nil RTOS semaphores source file.

```
#include "ch.h"
```

Functions

- msg_t chSemWaitTimeout (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- msg_t chSemWaitTimeoutS (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- void chSemSignal (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemSignalI (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemResetWithMessage (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
- void chSemResetWithMessageI (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.

8.11.1 Detailed Description

Nil RTOS semaphores source file.

8.12 chconf.h File Reference

Configuration file template.

Macros

- #define _CHIBIOS_NIL_CONF_
- #define _CHIBIOS_NIL_CONF_VER_4_0_

Kernel parameters and options

- #define CH_CFG_MAX_THREADS 4
  Maximum number of user threads in the application.
- #define CH_CFG_AUTOSTART_THREADS TRUE
  Auto starts threads when `chSysInit()` is invoked.

System timer settings

- #define CH_CFG_ST_RESOLUTION 32
  System time counter resolution.
- #define CH_CFG_ST_FREQUENCY 1000
  System tick frequency.

- #define CH_CFG_ST_TIMEDELTA 0
  Time delta constant for the tick-less mode.

Subsystem options

- #define CH_CFG_USE_WAITEXIT TRUE
  Threads synchronization APIs.
- #define CH_CFG_USE_SEMAPHORES TRUE
  Semaphores APIs.
- #define CH_CFG_USE_MUTEXES FALSE
  Mutexes APIs.
- #define CH_CFG_USE_EVENTS TRUE
  Events Flags APIs.
- #define CH_CFG_USE_MESSAGES TRUE
  Synchronous Messages APIs.

OSLIB options

- #define CH_CFG_USE_MAILBOXES TRUE
  Mailboxes APIs.
- #define CH_CFG_USE_MEMCORE TRUE
  Core Memory Manager APIs.
- #define CH_CFG_MEMCORE_SIZE 0
  Managed RAM size.
- #define CH_CFG_USE_HEAP TRUE
  Heap Allocator APIs.
- #define CH_CFG_USE_MEMPOOLS TRUE
  Memory Pools Allocator APIs.
- #define CH_CFG_USE_OBJ_FIFOS TRUE
  Objects FIFOs APIs.
- #define CH_CFG_USE_PIPES TRUE
  Pipes APIs.
- #define CH_CFG_USE_OBJ_CACHES TRUE
  Objects Caches APIs.
- #define CH_CFG_USE_DELEGATES TRUE
  Delegate threads APIs.
- #define CH_CFG_USE_JOBS TRUE
  Jobs Queues APIs.

Objects factory options

- #define CH_CFG_USE_FACTORY TRUE
  Objects Factory APIs.
- #define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8
  Maximum length for object names.
- #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE
  Enables the registry of generic objects.
- #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE
  Enables factory for generic buffers.
- #define CH_CFG_FACTORY_SEMAPHORES TRUE
  Enables factory for semaphores.
- #define CH_CFG_FACTORY_MAILBOXES TRUE
  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_OBJ_FIFOS TRUE
  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_PIPES TRUE
  Enables factory for Pipes.

Debug options

- #define CH_DBG_STATISTICS FALSE
  Debug option, kernel statistics.
- #define CH_DBG_SYSTEM_STATE_CHECK TRUE
  Debug option, system state check.
- #define CH_DBG_ENABLE_CHECKS TRUE
  Debug option, parameters checks.
- #define CH_DBG_ENABLE_ASSERTS TRUE
  System assertions.
- #define CH_DBG_ENABLE_STACK_CHECK TRUE
  Stack check.

Kernel hooks

- #define CH_CFG_SYSTEM_INIT_HOOK()
  System initialization hook.
- #define CH_CFG_THREAD_EXT_FIELDS /∗ Add threads custom fields here.∗/
  Threads descriptor structure extension.
- #define CH_CFG_THREAD_EXT_INIT_HOOK(tr)
  Threads initialization hook.
- #define CH_CFG_THREAD_EXIT_HOOK(tp)
  Threads finalization hook.
- #define CH_CFG_IDLE_ENTER_HOOK()
  Idle thread enter hook.
- #define CH_CFG_IDLE_LEAVE_HOOK()
  Idle thread leave hook.
- #define CH_CFG_SYSTEM_HALT_HOOK(reason)
  System halt hook.

8.12.1 Detailed Description

Configuration file template.

A copy of this file must be placed in each project directory, it contains the application specific kernel settings.

8.13 lib.dox File Reference

8.14 chbsem.h File Reference

Binary semaphores structures and macros.

Data Structures

- struct ch_binary_semaphore
  Binary semaphore type.

Macros

- #define __BSEMAPHORE_DATA(name, taken)

  Data part of a static semaphore initializer.
- #define BSEMAPHORE_DECL(name, taken)

  Static semaphore initializer.

Typedefs

- typedef struct ch_binary_semaphore binary_semaphore_t

  Binary semaphore type.

Functions

- static void chBSemObjectInit (binary_semaphore_t *bsp, bool taken)

  Initializes a binary semaphore.
- static msg_t chBSemWait (binary_semaphore_t *bsp)

  Wait operation on the binary semaphore.
- static msg_t chBSemWaitS (binary_semaphore_t *bsp)

  Wait operation on the binary semaphore.
- static msg_t chBSemWaitTimeoutS (binary_semaphore_t *bsp, sysinterval_t timeout)

  Wait operation on the binary semaphore.
- static msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, sysinterval_t timeout)

  Wait operation on the binary semaphore.
- static void chBSemResetI (binary_semaphore_t *bsp, bool taken)

  Reset operation on the binary semaphore.
- static void chBSemReset (binary_semaphore_t *bsp, bool taken)

  Reset operation on the binary semaphore.
- static void chBSemSignalI (binary_semaphore_t *bsp)

  Performs a signal operation on a binary semaphore.
- static void chBSemSignal (binary_semaphore_t *bsp)

  Performs a signal operation on a binary semaphore.
- static bool chBSemGetStateI (const binary_semaphore_t *bsp)

  Returns the binary semaphore current state.

8.14.1 Detailed Description

Binary semaphores structures and macros.

Binary semaphores related APIs and services.

Operation mode

Binary semaphores are implemented as a set of inline functions that use the existing counting semaphores primitives. The difference between counting and binary semaphores is that the counter of binary semaphores is not allowed to grow above the value 1. Repeated signal operation are ignored. A binary semaphore can thus have only two defined states:

- **Taken**, when its counter has a value of zero or lower than zero. A negative number represent the number of threads queued on the binary semaphore.
- **Not taken**, when its counter has a value of one.

Binary semaphores are different from mutexes because there is no concept of ownership, a binary semaphore can be taken by a thread and signaled by another thread or an interrupt handler, mutexes can only be taken and released by the same thread. Another difference is that binary semaphores, unlike mutexes, do not implement the priority inheritance protocol.
In order to use the binary semaphores APIs the `CH_CFG_USE_SEMAPHORES` option must be enabled in `chconf.h`.

8.15 chdelegates.h File Reference

Delegate threads macros and structures.

```
#include <stdarg.h>
```

Typedefs

- typedef msg_t(∗ delegate_veneer_t) (va_list ∗argsp)
  Type of a delegate veneer function.
- typedef msg_t(∗ delegate_fn0_t) (void)
  Type of a delegate function with no parameters.
- typedef msg_t(∗ delegate_fn1_t) (msg_t p1)
  Type of a delegate function with one parameter.
- typedef msg_t(∗ delegate_fn2_t) (msg_t p1, msg_t p2)
  Type of a delegate function with two parameters.
- typedef msg_t(∗ delegate_fn3_t) (msg_t p1, msg_t p2, msg_t p3)
  Type of a delegate function with three parameters.
- typedef msg_t(∗ delegate_fn4_t) (msg_t p1, msg_t p2, msg_t p3, msg_t p4)
  Type of a delegate function with four parameters.

Functions

- msg_t __ch_delegate_fn0 (va_list *argsp)
  Veneer for functions with no parameters.
- msg_t __ch_delegate_fn1 (va_list *argsp)
  Veneer for functions with one parameter.
- msg_t __ch_delegate_fn2 (va_list *argsp)
  Veneer for functions with two parameters.
- msg_t __ch_delegate_fn3 (va_list *argsp)
  Veneer for functions with three parameters.
- msg_t __ch_delegate_fn4 (va_list *argsp)
  Veneer for functions with four parameters.
- void chDelegateDispatch (void)
  Call messages dispatching.
- msg_t chDelegateDispatchTimeout (sysinterval_t timeout)
  Call messages dispatching with timeout.
- msg_t chDelegateCallVeneer (thread_t *tp, delegate_veneer_t veneer,...)
  Triggers a function call on a delegate thread.
- static msg_t chDelegateCallDirect0 (thread_t *tp, delegate_fn0_t func)
  Direct call to a function with no parameters.
- static msg_t chDelegateCallDirect1 (thread_t *tp, delegate_fn1_t func, msg_t p1)
  Direct call to a function with one parameter.
- static msg_t chDelegateCallDirect2 (thread_t *tp, delegate_fn2_t func, msg_t p1, msg_t p2)
  Direct call to a function with two parameters.
- static msg_t chDelegateCallDirect3 (thread_t *tp, delegate_fn3_t func, msg_t p1, msg_t p2, msg_t p3)
  Direct call to a function with three parameters.
- static msg_t chDelegateCallDirect4 (thread_t *tp, delegate_fn4_t func, msg_t p1, msg_t p2, msg_t p3, msg_t p4)
  Direct call to a function with four parameters.

8.15.1 Detailed Description

Delegate threads macros and structures.

8.16 chfactory.h File Reference

ChibiOS objects factory structures and macros.

Data Structures

- struct ch_dyn_element
  Type of a dynamic object list element.
- struct ch_dyn_list
  Type of a dynamic object list.
- struct ch_registered_static_object
  Type of a registered object.
- struct ch_dyn_object

Type of a dynamic buffer object.

- struct ch_dyn_semaphore

  Type of a dynamic semaphore.
- struct ch_dyn_mailbox

  Type of a dynamic buffer object.
- struct ch_dyn_objects_fifo

  Type of a dynamic buffer object.
- struct ch_dyn_pipe

  Type of a dynamic pipe object.
- struct ch_objects_factory

  Type of the factory main object.

Macros

- #define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8

  Maximum length for object names.
- #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

  Enables the registry of generic objects.
- #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

  Enables factory for generic buffers.
- #define CH_CFG_FACTORY_SEMAPHORES TRUE

  Enables factory for semaphores.
- #define CH_CFG_FACTORY_MAILBOXES TRUE

  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_PIPES TRUE

  Enables factory for Pipes.
- #define CH_CFG_FACTORY_SEMAPHORES FALSE

  Enables factory for semaphores.
- #define CH_CFG_FACTORY_MAILBOXES FALSE

  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_OBJ_FIFOS FALSE

  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_PIPES FALSE

  Enables factory for Pipes.
- #define CH_FACTORY_REQUIRES_POOLS
- #define CH_FACTORY_REQUIRES_HEAP

Typedefs

- typedef struct ch_dyn_element dyn_element_t

  Type of a dynamic object list element.
- typedef struct ch_dyn_list dyn_list_t

  Type of a dynamic object list.
- typedef struct ch_registered_static_object registered_object_t

  Type of a registered object.
- typedef struct ch_dyn_object dyn_buffer_t

Type of a dynamic buffer object.

- typedef struct ch_dyn_semaphore dyn_semaphore_t

  Type of a dynamic semaphore.
- typedef struct ch_dyn_mailbox dyn_mailbox_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_pipe dyn_pipe_t

  Type of a dynamic pipe object.
- typedef struct ch_objects_factory objects_factory_t

  Type of the factory main object.

Functions

- void __factory_init (void)

  Initializes the objects factory.
- registered_object_t * chFactoryRegisterObject (const char *name, void *objp)

  Registers a generic object.
- registered_object_t * chFactoryFindObject (const char *name)

  Retrieves a registered object.
- registered_object_t * chFactoryFindObjectByPointer (void *objp)

  Retrieves a registered object by pointer.
- void chFactoryReleaseObject (registered_object_t *rop)

  Releases a registered object.
- dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)

  Creates a generic dynamic buffer object.
- dyn_buffer_t * chFactoryFindBuffer (const char *name)

  Retrieves a dynamic buffer object.
- void chFactoryReleaseBuffer (dyn_buffer_t *dbp)

  Releases a dynamic buffer object.
- dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)

  Creates a dynamic semaphore object.
- dyn_semaphore_t * chFactoryFindSemaphore (const char *name)

  Retrieves a dynamic semaphore object.
- void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)

  Releases a dynamic semaphore object.
- dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)

  Creates a dynamic mailbox object.
- dyn_mailbox_t * chFactoryFindMailbox (const char *name)

  Retrieves a dynamic mailbox object.
- void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)

  Releases a dynamic mailbox object.
- dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)

  Creates a dynamic "objects FIFO" object.
- dyn_objects_fifo_t * chFactoryFindObjectsFIFO (const char *name)

  Retrieves a dynamic "objects FIFO" object.
- void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)

  Releases a dynamic "objects FIFO" object.
- dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)

Creates a dynamic pipe object.

- dyn_pipe_t * chFactoryFindPipe (const char *name)

  Retrieves a dynamic pipe object.
- void chFactoryReleasePipe (dyn_pipe_t *dpp)

  Releases a dynamic pipe object.
- static dyn_element_t * chFactoryDuplicateReference (dyn_element_t *dep)

  Duplicates an object reference.
- static void * chFactoryGetObject (registered_object_t *rop)

  Returns the pointer to the inner registered object.
- static size_t chFactoryGetBufferSize (dyn_buffer_t *dbp)

  Returns the size of a generic dynamic buffer object.
- static uint8_t * chFactoryGetBuffer (dyn_buffer_t *dbp)

  Returns the pointer to the inner buffer.
- static semaphore_t * chFactoryGetSemaphore (dyn_semaphore_t *dsp)

  Returns the pointer to the inner semaphore.
- static mailbox_t * chFactoryGetMailbox (dyn_mailbox_t *dmp)

  Returns the pointer to the inner mailbox.
- static objects_fifo_t * chFactoryGetObjectsFIFO (dyn_objects_fifo_t *dofp)

  Returns the pointer to the inner objects FIFO.
- static pipe_t * chFactoryGetPipe (dyn_pipe_t *dpp)

  Returns the pointer to the inner pipe.

8.16.1 Detailed Description

ChibiOS objects factory structures and macros.

8.17 chjobs.h File Reference

Jobs Queues structures and macros.

Data Structures

- struct ch_jobs_queue

  Type of a jobs queue.
- struct ch_job_descriptor

  Type of a job descriptor.

Macros

- #define MSG_JOB_NULL ((msg_t)-2)

  Dispatcher return code in case of a `JOB_NUL` has been received.

Typedefs

- typedef struct ch_jobs_queue jobs_queue_t
  Type of a jobs queue.
- typedef void(* job_function_t) (void *arg)
  Type of a job function.
- typedef struct ch_job_descriptor job_descriptor_t
  Type of a job descriptor.

Functions

- static void chJobObjectInit (jobs_queue_t *jqp, size_t jobsn, job_descriptor_t *jobsbuf, msg_t *msgbuf)
  Initializes a jobs queue object.
- static job_descriptor_t * chJobGet (jobs_queue_t *jqp)
  Allocates a free job object.
- static job_descriptor_t * chJobGetI (jobs_queue_t *jqp)
  Allocates a free job object.
- static job_descriptor_t * chJobGetTimeoutS (jobs_queue_t *jqp, sysinterval_t timeout)
  Allocates a free job object.
- static job_descriptor_t * chJobGetTimeout (jobs_queue_t *jqp, sysinterval_t timeout)
  Allocates a free job object.
- static void chJobPostI (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts a job object.
- static void chJobPostS (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts a job object.
- static void chJobPost (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts a job object.
- static void chJobPostAheadI (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts an high priority job object.
- static void chJobPostAheadS (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts an high priority job object.
- static void chJobPostAhead (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts an high priority job object.
- static msg_t chJobDispatch (jobs_queue_t *jqp)
  Waits for a job then executes it.
- static msg_t chJobDispatchTimeout (jobs_queue_t *jqp, sysinterval_t timeout)
  Waits for a job then executes it.

8.17.1 Detailed Description

Jobs Queues structures and macros.

This module implements queues of generic jobs to be delegated asynchronously to a pool of dedicated threads. Operations defined for Jobs Queues

- **Get**: An job object is taken from the pool of the available jobs.
- **Post**: A job is posted to the queue, it will be returned to the pool after execution.

8.18 chlib.h File Reference

ChibiOS/LIB main include file.

```
#include "chbsem.h"
#include "chmboxes.h"
#include "chmemcore.h"
#include "chmemheaps.h"
#include "chmempools.h"
#include "chobjfifos.h"
#include "chpipes.h"
#include "chobjcaches.h"
#include "chdelegates.h"
#include "chjobs.h"
#include "chfactory.h"
```

Macros

- #define __CHIBIOS_OSLIB__

  ChibiOS/LIB identification macro.
- #define CH_OSLIB_STABLE 1

  Stable release flag.
- #define CH_CFG_USE_FACTORY FALSE
- #define CH_CFG_USE_HEAP FALSE
- #define CH_CFG_USE_MEMPOOLS FALSE
- #define CH_CFG_USE_OBJ_FIFOS FALSE
- #define CH_CFG_USE_PIPES FALSE
- #define CH_CFG_USE_OBJ_CACHES FALSE
- #define CH_CFG_USE_DELEGATES FALSE
- #define CH_CFG_USE_JOBS FALSE
- #define CH_CFG_USE_MAILBOXES FALSE

ChibiOS/LIB version identification

- #define CH_OSLIB_VERSION "1.3.2"

  OS Library version string.
- #define CH_OSLIB_MAJOR 1

  OS Library version major number.
- #define CH_OSLIB_MINOR 3

  OS Library version minor number.
- #define CH_OSLIB_PATCH 2

  OS Library version patch number.

Functions

- static void __oslib_init (void)

  Initialization of all library modules.

8.18.1 Detailed Description

ChibiOS/LIB main include file.

This header includes all the required library headers. This file is meant do be included by `ch.h` not directly by user.

8.19 chmboxes.h File Reference

Mailboxes macros and structures.

Data Structures

- struct mailbox_t

  Structure representing a mailbox object.

Macros

- #define __MAILBOX_DATA(name, buffer, size)

  Data part of a static mailbox initializer.
- #define MAILBOX_DECL(name, buffer, size)

  Static mailbox initializer.

Functions

- void chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)

  Initializes a `mailbox_t` object.
- void chMBReset (mailbox_t *mbp)

  Resets a `mailbox_t` object.
- void chMBResetI (mailbox_t *mbp)

  Resets a `mailbox_t` object.
- msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts a message into a mailbox.
- msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts a message into a mailbox.
- msg_t chMBPostI (mailbox_t *mbp, msg_t msg)

  Posts a message into a mailbox.
- msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)

  Posts an high priority message into a mailbox.
- msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)

  Retrieves a message from a mailbox.
- msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)

  Retrieves a message from a mailbox.

- msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)
  Retrieves a message from a mailbox.
- static size_t chMBGetSizeI (const mailbox_t *mbp)
  Returns the mailbox buffer size as number of messages.
- static size_t chMBGetUsedCountI (const mailbox_t *mbp)
  Returns the number of used message slots into a mailbox.
- static size_t chMBGetFreeCountI (const mailbox_t *mbp)
  Returns the number of free message slots into a mailbox.
- static msg_t chMBPeekI (const mailbox_t *mbp)
  Returns the next message in the queue without removing it.
- static void chMBResumeX (mailbox_t *mbp)
  Terminates the reset state.

8.19.1 Detailed Description

Mailboxes macros and structures.

8.20 chmemcore.h File Reference

Core memory manager macros and structures.

Data Structures

- struct memcore_t
  Type of memory core object.

Macros

- #define CH_CFG_MEMCORE_SIZE 0
  Managed RAM size.
- #define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI
  Allocates a memory block.
- #define chCoreAllocAlignedWithOffset chCoreAllocFromTop
  Allocates a memory block.

Typedefs

- typedef void *(* memgetfunc_t) (size_t size, unsigned align)
  Memory get function.
- typedef void *(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)
  Enhanced memory get function.

Functions

- void __core_init (void)
  Low level memory manager initialization.
- void * chCoreAllocFromBaseI (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTopI (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the top address downward.
- void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the top address downward.
- size_t chCoreGetStatusX (void)
  Core memory status.
- static void * chCoreAllocAlignedI (size_t size, unsigned align)
  Allocates a memory block.
- static void * chCoreAllocAligned (size_t size, unsigned align)
  Allocates a memory block.
- static void * chCoreAllocI (size_t size)
  Allocates a memory block.
- static void * chCoreAlloc (size_t size)
  Allocates a memory block.

8.20.1 Detailed Description

Core memory manager macros and structures.

8.21 chmemheaps.h File Reference

Memory heaps macros and structures.

Data Structures

- union heap_header
  Memory heap block header.
- struct memory_heap
  Structure describing a memory heap.

Macros

- #define CH_HEAP_ALIGNMENT 8U
  Minimum alignment used for heap.
- #define CH_HEAP_AREA(name, size)
  Allocation of an aligned static heap buffer.

Typedefs

- typedef struct memory_heap memory_heap_t

  Type of a memory heap.
- typedef union heap_header heap_header_t

  Type of a memory heap header.

Functions

- void __heap_init (void)

  Initializes the default heap.
- void chHeapObjectInit (memory_heap_t *heapp, void *buf, size_t size)

  Initializes a memory heap from a static memory area.
- void * chHeapAllocAligned (memory_heap_t *heapp, size_t size, unsigned align)

  Allocates a block of memory from the heap by using the first-fit algorithm.
- void chHeapFree (void *p)

  Frees a previously allocated memory block.
- size_t chHeapStatus (memory_heap_t *heapp, size_t *totalp, size_t *largestp)

  Reports the heap status.
- static void * chHeapAlloc (memory_heap_t *heapp, size_t size)

  Allocates a block of memory from the heap by using the first-fit algorithm.
- static size_t chHeapGetSize (const void *p)

  Returns the size of an allocated block.

8.21.1 Detailed Description

Memory heaps macros and structures.

8.22 chmempools.h File Reference

Memory Pools macros and structures.

Data Structures

- struct pool_header

  Memory pool free object header.
- struct memory_pool_t

  Memory pool descriptor.
- struct guarded_memory_pool_t

  Guarded memory pool descriptor.

Macros

- #define __MEMORYPOOL_DATA(name, size, align, provider)

  Data part of a static memory pool initializer.
- #define MEMORYPOOL_DECL(name, size, align, provider)

  Static memory pool initializer.
- #define __GUARDEDMEMORYPOOL_DATA(name, size, align)

  Data part of a static guarded memory pool initializer.
- #define GUARDEDMEMORYPOOL_DECL(name, size, align)

  Static guarded memory pool initializer.

Functions

- void chPoolObjectInitAligned (memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)

  Initializes an empty memory pool.
- void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)

  Loads a memory pool with an array of static objects.
- void * chPoolAllocI (memory_pool_t *mp)

  Allocates an object from a memory pool.
- void * chPoolAlloc (memory_pool_t *mp)

  Allocates an object from a memory pool.
- void chPoolFreeI (memory_pool_t *mp, void *objp)

  Releases an object into a memory pool.
- void chPoolFree (memory_pool_t *mp, void *objp)

  Releases an object into a memory pool.
- void chGuardedPoolObjectInitAligned (guarded_memory_pool_t *gmp, size_t size, unsigned align)

  Initializes an empty guarded memory pool.
- void chGuardedPoolLoadArray (guarded_memory_pool_t *gmp, void *p, size_t n)

  Loads a guarded memory pool with an array of static objects.
- void * chGuardedPoolAllocTimeoutS (guarded_memory_pool_t *gmp, sysinterval_t timeout)

  Allocates an object from a guarded memory pool.
- void * chGuardedPoolAllocTimeout (guarded_memory_pool_t *gmp, sysinterval_t timeout)

  Allocates an object from a guarded memory pool.
- void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)

  Releases an object into a guarded memory pool.
- static void chPoolObjectInit (memory_pool_t *mp, size_t size, memgetfunc_t provider)

  Initializes an empty memory pool.
- static void chPoolAdd (memory_pool_t *mp, void *objp)

  Adds an object to a memory pool.
- static void chPoolAddI (memory_pool_t *mp, void *objp)

  Adds an object to a memory pool.
- static void chGuardedPoolObjectInit (guarded_memory_pool_t *gmp, size_t size)

  Initializes an empty guarded memory pool.
- static cnt_t chGuardedPoolGetCounterI (guarded_memory_pool_t *gmp)

  Gets the count of objects in a guarded memory pool.
- static void * chGuardedPoolAllocI (guarded_memory_pool_t *gmp)

  Allocates an object from a guarded memory pool.
- static void chGuardedPoolFreeI (guarded_memory_pool_t *gmp, void *objp)

  Releases an object into a guarded memory pool.
- static void chGuardedPoolFreeS (guarded_memory_pool_t *gmp, void *objp)

Releases an object into a guarded memory pool.

- static void chGuardedPoolAdd (guarded_memory_pool_t *gmp, void *objp)

  Adds an object to a guarded memory pool.
- static void chGuardedPoolAddI (guarded_memory_pool_t *gmp, void *objp)

  Adds an object to a guarded memory pool.
- static void chGuardedPoolAddS (guarded_memory_pool_t *gmp, void *objp)

  Adds an object to a guarded memory pool.

8.22.1 Detailed Description

Memory Pools macros and structures.

8.23 chobjcaches.h File Reference

Objects Caches macros and structures.

Data Structures

- struct ch_oc_hash_header

  Structure representing an hash table element.
- struct ch_oc_lru_header

  Structure representing an hash table element.
- struct ch_oc_object

  Structure representing a cached object.
- struct ch_objects_cache

  Structure representing a cache object.

Macros

Cached objects flags

- #define OC_FLAG_INLRU 0x00000001U
- #define OC_FLAG_INHASH 0x00000002U
- #define OC_FLAG_SHARED 0x00000004U
- #define OC_FLAG_NOTSYNC 0x00000008U
- #define OC_FLAG_LAZYWRITE 0x00000010U
- #define OC_FLAG_FORGET 0x00000020U

Typedefs

- typedef uint32_t oc_flags_t

  Flags of cached objects.
- typedef struct ch_oc_hash_header oc_hash_header_t

  Type of an hash element header.
- typedef struct ch_oc_lru_header oc_lru_header_t

  Type of an LRU element header.
- typedef struct ch_oc_object oc_object_t

  Type of a cached object.
- typedef struct ch_objects_cache objects_cache_t

  Type of a cache object.
- typedef bool(* oc_readf_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)

  Object read function.
- typedef bool(* oc_writef_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)

  Object write function.

Functions

- void chCacheObjectInit (objects_cache_t *ocp, ucnt_t hashn, oc_hash_header_t *hashp, ucnt_t objn, size_t objsz, void *objvp, oc_readf_t readf, oc_writef_t writef)

  Initializes a `objects_cache_t` object.
- oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)

  Retrieves an object from the cache.
- void chCacheReleaseObjectI (objects_cache_t *ocp, oc_object_t *objp)

  Releases an object into the cache.
- bool chCacheReadObject (objects_cache_t *ocp, oc_object_t *objp, bool async)

  Reads object data from the storage.
- bool chCacheWriteObject (objects_cache_t *ocp, oc_object_t *objp, bool async)

  Writes the object data back to storage.
- static void chCacheReleaseObject (objects_cache_t *ocp, oc_object_t *objp)

  Releases an object into the cache.

8.23.1 Detailed Description

Objects Caches macros and structures.

8.24 chobjfifos.h File Reference

Objects FIFO structures and macros.

Data Structures

- struct ch_objects_fifo

  Type of an objects FIFO.

Typedefs

- typedef struct ch_objects_fifo objects_fifo_t

  Type of an objects FIFO.

Functions

- static void chFifoObjectInitAligned (objects_fifo_t *ofp, size_t objsize, size_t objn, unsigned objalign, void *objbuf, msg_t *msgbuf)

  Initializes a FIFO object.
- static void chFifoObjectInit (objects_fifo_t *ofp, size_t objsize, size_t objn, void *objbuf, msg_t *msgbuf)

  Initializes a FIFO object.
- static void * chFifoTakeObjectI (objects_fifo_t *ofp)

  Allocates a free object.
- static void * chFifoTakeObjectTimeoutS (objects_fifo_t *ofp, sysinterval_t timeout)

  Allocates a free object.
- static void * chFifoTakeObjectTimeout (objects_fifo_t *ofp, sysinterval_t timeout)

  Allocates a free object.

- static void chFifoReturnObjectI (objects_fifo_t *ofp, void *objp)
  Releases a fetched object.
- static void chFifoReturnObjectS (objects_fifo_t *ofp, void *objp)
  Releases a fetched object.
- static void chFifoReturnObject (objects_fifo_t *ofp, void *objp)
  Releases a fetched object.
- static void chFifoSendObjectI (objects_fifo_t *ofp, void *objp)
  Posts an object.
- static void chFifoSendObjectS (objects_fifo_t *ofp, void *objp)
  Posts an object.
- static void chFifoSendObject (objects_fifo_t *ofp, void *objp)
  Posts an object.
- static void chFifoSendObjectAheadI (objects_fifo_t *ofp, void *objp)
  Posts an high priority object.
- static void chFifoSendObjectAheadS (objects_fifo_t *ofp, void *objp)
  Posts an high priority object.
- static void chFifoSendObjectAhead (objects_fifo_t *ofp, void *objp)
  Posts an high priority object.
- static msg_t chFifoReceiveObjectI (objects_fifo_t *ofp, void **objpp)
  Fetches an object.
- static msg_t chFifoReceiveObjectTimeoutS (objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)
  Fetches an object.
- static msg_t chFifoReceiveObjectTimeout (objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)
  Fetches an object.

8.24.1 Detailed Description

Objects FIFO structures and macros.

This module implements a generic FIFO queue of objects by coupling a Guarded Memory Pool (for objects storage) and a MailBox.
On the sender side free objects are taken from the pool, filled and then sent to the receiver, on the receiver side objects are fetched, used and then returned to the pool. Operations defined for object FIFOs:

- **Take**: An object is taken from the pool of the free objects, can be blocking.
- **Return**: An object is returned to the pool of the free objects, it is guaranteed to be non-blocking.
- **Send**: An object is sent through the mailbox, it is guaranteed to be non-blocking
- **Receive**: An object is received from the mailbox, can be blocking.

8.25 chpipes.h File Reference

Pipes macros and structures.

Data Structures

- struct pipe_t
  Structure representing a pipe object.

Macros

- #define __PIPE_DATA(name, buffer, size)

  Data part of a static pipe initializer.
- #define PIPE_DECL(name, buffer, size)

  Static pipe initializer.

Functions

- void chPipeObjectInit (pipe_t *pp, uint8_t *buf, size_t n)

  Initializes a `mailbox_t` object.
- void chPipeReset (pipe_t *pp)

  Resets a `pipe_t` object.
- size_t chPipeWriteTimeout (pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe write with timeout.
- size_t chPipeReadTimeout (pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe read with timeout.
- static size_t chPipeGetSize (const pipe_t *pp)

  Returns the pipe buffer size as number of bytes.
- static size_t chPipeGetUsedCount (const pipe_t *pp)

  Returns the number of used byte slots into a pipe.
- static size_t chPipeGetFreeCount (const pipe_t *pp)

  Returns the number of free byte slots into a pipe.
- static void chPipeResume (pipe_t *pp)

  Terminates the reset state.

8.25.1 Detailed Description

Pipes macros and structures.

8.26 chdelegates.c File Reference

Delegate threads code.

```
#include "ch.h"
```

Functions

- msg_t __ch_delegate_fn0 (va_list *argsp)
  Veneer for functions with no parameters.
- msg_t __ch_delegate_fn1 (va_list *argsp)
  Veneer for functions with one parameter.
- msg_t __ch_delegate_fn2 (va_list *argsp)
  Veneer for functions with two parameters.
- msg_t __ch_delegate_fn3 (va_list *argsp)
  Veneer for functions with three parameters.
- msg_t __ch_delegate_fn4 (va_list *argsp)
  Veneer for functions with four parameters.
- msg_t chDelegateCallVeneer (thread_t *tp, delegate_veneer_t veneer,...)
  Triggers a function call on a delegate thread.
- void chDelegateDispatch (void)
  Call messages dispatching.
- msg_t chDelegateDispatchTimeout (sysinterval_t timeout)
  Call messages dispatching with timeout.

8.26.1 Detailed Description

Delegate threads code.

Delegate threads.

Operation mode

A delegate thread is a thread performing function calls triggered by other threads. This functionality is especially useful when encapsulating a library not designed for threading into a delegate thread. Other threads have access to the library without having to worry about mutual exclusion.

Precondition

In order to use the pipes APIs the `CH_CFG_USE_DELEGATES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

8.27 chfactory.c File Reference

ChibiOS objects factory and registry code.

```
#include <string.h>
#include "ch.h"
```

Macros

- #define F_LOCK()
- #define F_UNLOCK()

Functions

- static void copy_name (const char *sp, char *dp)
- static void dyn_list_init (dyn_list_t *dlp)
- static dyn_element_t * dyn_list_find (const char *name, dyn_list_t *dlp)
- static dyn_element_t * dyn_list_find_prev (dyn_element_t *element, dyn_list_t *dlp)
- static dyn_element_t * dyn_list_unlink (dyn_element_t *prev)
- static dyn_element_t * dyn_create_object_heap (const char *name, dyn_list_t *dlp, size_t size, unsigned align)
- static void dyn_release_object_heap (dyn_element_t *dep, dyn_list_t *dlp)
- static dyn_element_t * dyn_create_object_pool (const char *name, dyn_list_t *dlp, memory_pool_t *mp)
- static void dyn_release_object_pool (dyn_element_t *dep, dyn_list_t *dlp, memory_pool_t *mp)
- static dyn_element_t * dyn_find_object (const char *name, dyn_list_t *dlp)
- void __factory_init (void)

  Initializes the objects factory.
- registered_object_t * chFactoryRegisterObject (const char *name, void *objp)

  Registers a generic object.
- registered_object_t * chFactoryFindObject (const char *name)

  Retrieves a registered object.
- registered_object_t * chFactoryFindObjectByPointer (void *objp)

  Retrieves a registered object by pointer.
- void chFactoryReleaseObject (registered_object_t *rop)

  Releases a registered object.
- dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)

  Creates a generic dynamic buffer object.
- dyn_buffer_t * chFactoryFindBuffer (const char *name)

  Retrieves a dynamic buffer object.
- void chFactoryReleaseBuffer (dyn_buffer_t *dbp)

  Releases a dynamic buffer object.
- dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)

  Creates a dynamic semaphore object.
- dyn_semaphore_t * chFactoryFindSemaphore (const char *name)

  Retrieves a dynamic semaphore object.
- void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)

  Releases a dynamic semaphore object.
- dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)

  Creates a dynamic mailbox object.
- dyn_mailbox_t * chFactoryFindMailbox (const char *name)

  Retrieves a dynamic mailbox object.
- void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)

  Releases a dynamic mailbox object.
- dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)

  Creates a dynamic "objects FIFO" object.
- dyn_objects_fifo_t * chFactoryFindObjectsFIFO (const char *name)

  Retrieves a dynamic "objects FIFO" object.

- void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)
  Releases a dynamic "objects FIFO" object.
- dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)
  Creates a dynamic pipe object.
- dyn_pipe_t * chFactoryFindPipe (const char *name)
  Retrieves a dynamic pipe object.
- void chFactoryReleasePipe (dyn_pipe_t *dpp)
  Releases a dynamic pipe object.

Variables

- objects_factory_t ch_factory
  Factory object static instance.

8.27.1 Detailed Description

ChibiOS objects factory and registry code.

8.28 chmboxes.c File Reference

Mailboxes code.

```
#include "ch.h"
```

Functions

- void chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)
  Initializes a `mailbox_t` object.
- void chMBReset (mailbox_t *mbp)
  Resets a `mailbox_t` object.
- void chMBResetI (mailbox_t *mbp)
  Resets a `mailbox_t` object.
- msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  Posts a message into a mailbox.
- msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  Posts a message into a mailbox.
- msg_t chMBPostI (mailbox_t *mbp, msg_t msg)
  Posts a message into a mailbox.
- msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)
  Posts an high priority message into a mailbox.
- msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  Retrieves a message from a mailbox.
- msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  Retrieves a message from a mailbox.
- msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)
  Retrieves a message from a mailbox.

8.28.1 Detailed Description

Mailboxes code.

8.29 chmemcore.c File Reference

Core memory manager code.

```
#include "ch.h"
```

Functions

- void __core_init (void)
  Low level memory manager initialization.
- void * chCoreAllocFromBaseI (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTopI (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the top address downward.
- void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the top address downward.
- size_t chCoreGetStatusX (void)
  Core memory status.

Variables

- memcore_t ch_memcore
  Memory core descriptor.

8.29.1 Detailed Description

Core memory manager code.

8.30 chmemheaps.c File Reference

Memory heaps code.

```
#include "ch.h"
```

Macros

- #define H_LOCK(h)
- #define H_UNLOCK(h)
- #define H_BLOCK(hp)
- #define H_LIMIT(hp)
- #define H_NEXT(hp)
- #define H_PAGES(hp)
- #define H_HEAP(hp)
- #define H_SIZE(hp)
- #define NPAGES(p1, p2)

Functions

- void __heap_init (void)

  Initializes the default heap.
- void chHeapObjectInit (memory_heap_t *heapp, void *buf, size_t size)

  Initializes a memory heap from a static memory area.
- void * chHeapAllocAligned (memory_heap_t *heapp, size_t size, unsigned align)

  Allocates a block of memory from the heap by using the first-fit algorithm.
- void chHeapFree (void *p)

  Frees a previously allocated memory block.
- size_t chHeapStatus (memory_heap_t *heapp, size_t *totalp, size_t *largestp)

  Reports the heap status.

Variables

- static memory_heap_t default_heap

  Default heap descriptor.

8.30.1 Detailed Description

Memory heaps code.

8.31 chmempools.c File Reference

Memory Pools code.

```
#include "ch.h"
```

Functions

- void chPoolObjectInitAligned (memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)
  Initializes an empty memory pool.
- void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)
  Loads a memory pool with an array of static objects.
- void * chPoolAllocI (memory_pool_t *mp)
  Allocates an object from a memory pool.
- void * chPoolAlloc (memory_pool_t *mp)
  Allocates an object from a memory pool.
- void chPoolFreeI (memory_pool_t *mp, void *objp)
  Releases an object into a memory pool.
- void chPoolFree (memory_pool_t *mp, void *objp)
  Releases an object into a memory pool.
- void chGuardedPoolObjectInitAligned (guarded_memory_pool_t *gmp, size_t size, unsigned align)
  Initializes an empty guarded memory pool.
- void chGuardedPoolLoadArray (guarded_memory_pool_t *gmp, void *p, size_t n)
  Loads a guarded memory pool with an array of static objects.
- void * chGuardedPoolAllocTimeoutS (guarded_memory_pool_t *gmp, sysinterval_t timeout)
  Allocates an object from a guarded memory pool.
- void * chGuardedPoolAllocTimeout (guarded_memory_pool_t *gmp, sysinterval_t timeout)
  Allocates an object from a guarded memory pool.
- void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)
  Releases an object into a guarded memory pool.

8.31.1 Detailed Description

Memory Pools code.

8.32 chobjcaches.c File Reference

Objects Caches code.

```
#include "ch.h"
```

Macros

- #define OC_HASH_FUNCTION(ocp, group, key)
- #define HASH_INSERT(ocp, objp, group, key)
- #define HASH_REMOVE(objp)
- #define LRU_INSERT_HEAD(ocp, objp)
- #define LRU_INSERT_TAIL(ocp, objp)
- #define LRU_REMOVE(objp)

Functions

- static oc_object_t * hash_get_s (objects_cache_t *ocp, uint32_t group, uint32_t key)

  Returns an object pointer from the cache, if present.
- static oc_object_t * lru_get_last_s (objects_cache_t *ocp)

  Gets the least recently used object buffer from the LRU list.
- void chCacheObjectInit (objects_cache_t *ocp, ucnt_t hashn, oc_hash_header_t *hashp, ucnt_t objn, size_t objsz, void *objvp, oc_readf_t readf, oc_writef_t writef)

  Initializes a `objects_cache_t` object.
- oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)

  Retrieves an object from the cache.
- void chCacheReleaseObjectI (objects_cache_t *ocp, oc_object_t *objp)

  Releases an object into the cache.
- bool chCacheReadObject (objects_cache_t *ocp, oc_object_t *objp, bool async)

  Reads object data from the storage.
- bool chCacheWriteObject (objects_cache_t *ocp, oc_object_t *objp, bool async)

  Writes the object data back to storage.

8.32.1 Detailed Description

Objects Caches code.

Objects caches.

Operation mode

An object cache allows to retrieve and release objects from a slow media, for example a disk or flash.
The most recently used objects are kept in a series of RAM buffers making access faster. Objects are identified by a pair <group, key> which could be mapped, for example, to a disk drive identifier and sector identifier.
Read and write operations are performed using externally-supplied functions, the cache is device-agnostic.
The cache uses internally an hash table, the size of the table should be dimensioned to minimize the risk of hash collisions, a factor of two is usually acceptable, it depends on the specific application requirements.
Operations defined for caches:

- **Get Object**: Retrieves an object from cache, if not present then an empty buffer is returned.
- **Read Object**: Retrieves an object from cache, if not present a buffer is allocated and the object is read from the media.
- **Release Object**: Releases an object to the cache handling the media update, if required.

Precondition

In order to use the pipes APIs the `CH_CFG_USE_OBJ_CACHES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

8.33 chpipes.c File Reference

Pipes code.

```
#include <string.h>
#include "ch.h"
```

Macros

- #define PC_INIT(p)
- #define PC_LOCK(p)
- #define PC_UNLOCK(p)
- #define PW_INIT(p)
- #define PW_LOCK(p)
- #define PW_UNLOCK(p)
- #define PR_INIT(p)
- #define PR_LOCK(p)
- #define PR_UNLOCK(p)

Functions

- static size_t pipe_write (pipe_t *pp, const uint8_t *bp, size_t n)
  Non-blocking pipe write.
- static size_t pipe_read (pipe_t *pp, uint8_t *bp, size_t n)
  Non-blocking pipe read.
- void chPipeObjectInit (pipe_t *pp, uint8_t *buf, size_t n)
  Initializes a `mailbox_t` object.
- void chPipeReset (pipe_t *pp)
  Resets a `pipe_t` object.
- size_t chPipeWriteTimeout (pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)
  Pipe write with timeout.
- size_t chPipeReadTimeout (pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)
  Pipe read with timeout.

8.33.1 Detailed Description

Pipes code.

Byte pipes.

Operation mode

A pipe is an asynchronous communication mechanism.
Operations defined for mailboxes:

- **Write**: Writes a buffer of data in the pipe in FIFO order.
- **Read**: A buffer of data is read from the read and removed.
- **Reset**: The pipe is emptied and all the stored data is lost.

Precondition

In order to use the pipes APIs the `CH_CFG_USE_PIPES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

Index